國立交通大學

資訊學院　資訊學程

碩 士 論 文

類神經網路與基因演算法於井測資料反推

Neural Networks and Genetic Algorithms for Well Logging Inversion

研 究 生：余勝棟

指導教授：黃國源　教授

中 華 民 國 九 十 七 年 七 月

類神經網路與基因演算法於井測資料反推
Neural Networks and Genetic Algorithms for Well Logging Inversion

研　究　生：余勝棟　　　　　Student：Sheng-Tung Yu

指導教授：黃國源　博士　　　Advisor：Dr. Kou-Yuan Huang

國　立　交　通　大　學

資訊學院　　資訊學程

碩　士　論　文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

July 2008

Hsinchu, Taiwan

中華民國九十七年七月

類神經網路與基因演算法於井測資料反推

學生:余勝棟　　　　　　　　　　　　指導教授:黃國源 博士

國 立 交 通 大 學　　資 訊 學 院　　資 訊 學 程 碩 士 班

摘　　要

在井測資料的反推問題中，井測合成資料與地層真實導電率之間是一種非線性的映對關係，而類神經網路能夠避免複雜的理論計算，經由訓練過程來調整權重值，以逼近輸入輸出值之間的映對關係，因此本研究以類神經網路為基礎，發展高階特徵值類神經網路架構，以應用於井測資料的反推。

由於常用以訓練類神經網路的梯度下降法，有易於陷入區域最小值的缺點，因此，我們以結合基因演算法來改善網路學習的效率。另外，梯度下降法的收斂較為緩慢，因此使用共軛梯度法來增進學習速率。為了加強網路的非線性映對能力，我們提出高階特徵值類神經網路，其特性為使用次方函數來增加輸入特徵值的高階項。此外，為了提高訓練樣本的數量以增進收斂效率，我們測試了不同輸入神經元數量的網路架構。另外，經由實驗結果比較得知，含有 1 層隱藏層的網路比不含隱藏層的網路有較好的收斂效果，因此我們採用含有 1 層隱藏層的網路。研究資料一共是 31 口井的井測合成資料，每一口井含有 200 點的輸入與期望輸出值資料。實驗以網路的實際輸出值與期望輸出值之間的平均絕對值誤差來評估網路的效能，以 Leave-one-out 的方式做 31 次試驗，每次試驗用 30 組資料做訓練，訓練完成的網路以剩下的 1 組資料做測試，經過 31 次的試驗，取其平均值做為該次實驗的結果。

為了驗證高階特徵值類神經網路的有效性，我們以合成的資料作為輸入特徵值來訓練網路，採用的網路架構為 30-36-10 (不含 bias)，使用共軛梯度法為訓練法則，網路訓練完成後輸入真實的井測資料，以測試網路的反推結果。實驗顯示，我們所提出的高階特徵值類神經網路，能夠有效的應用於井測資料的反推問題。在以類神經網路應用於井測資料的反推問題上，我們的研究結果提供了一個良好的網路架構。

關鍵字:類神經網路，基因演算法，井測資料反推。

Neural Networks and Genetic Algorithms for Well Logging Inversion

Student：Sheng-Tung Yu　　　　　Advisor：Dr. Kou-Yuan Huang

Degree Program of Computer Science
National Chiao Tung University

ABSTRACT

In well logging inversion problem, a non-linear mapping exists between the synthetic logging measurements and the true formation conductivity. Without complexity of theoretic computation, neural network is able to approximate the input-output mapping through training with the iterative adjustment of connection weights. In our study, we develop the higher-order feature neural nets on the basis of neural network, and then apply on well logging inversion.

The usually used training algorithm for neural network is gradient descent, which is easy to get trapped at local minimum, so we adopt a method that combine with genetic algorithm to improve the training efficiency. In addition, the convergence of gradient descent is slow, so we adopt the conjugate gradient to speed up the convergence. In order to make network more non-linear, we proposed higher-order feature neural nets that use functions to expand the input feature to higher degree. In order to use more training patterns and increase the convergence efficiency, we test various network architectures that use different number of input nodes. Besides, the experimental results show that the convergence efficiency of the network with 1 hidden layer is better than that without hidden layer, so we adopt the network with 1 hidden layer. We use 31 synthetic logging datasets. Each has 200 input features and corresponding outputs. The performance of network is evaluated by comparing the mean absolute error between the actual outputs and desired outputs. Leave-one-out validation method is used in experiments. Each time 30 datasets are used in training, the trained network is then tested with the left 1 dataset. After 31 trials, the network performance is computed by averaging these testing results.

To validate the effectiveness of higher-order feature neural nets, the network size is 30-36-10 (not include bias), we train the network using conjugate gradient with synthetic logging datasets, and the trained network is then tested with real field logs. Results obtained from our experiments have shown that the proposed higher-order feature neural nets can be used effectively to process the well logging inversion. Our study shows an effective architecture of neural network to apply on well logging data inversion.

Keywords: Neural network, genetic algorithm, well logging inversion.

誌　　謝

感謝恩師黃國源博士，一年多以來不厭其煩地給予學生指導，使學生不僅在學業上得到許多啟發，對於老師治學的嚴謹態度與做研究實事求是的精神，更是學生不管在求學上與工作上所積極學習的典範，令我受益良多。

感謝劉長遠教授、王榮華教授、以及蘇豐文教授，在口試期間給予學生修改論文上的諸多寶貴建議。

感謝實驗室的同學們，在這一段時間以來所給予我的鼓勵與幫助。最後感謝家人朋友以及所有關心我的人，謝謝你們陪我一路走來。

Contents

List of Tables

List of Figures

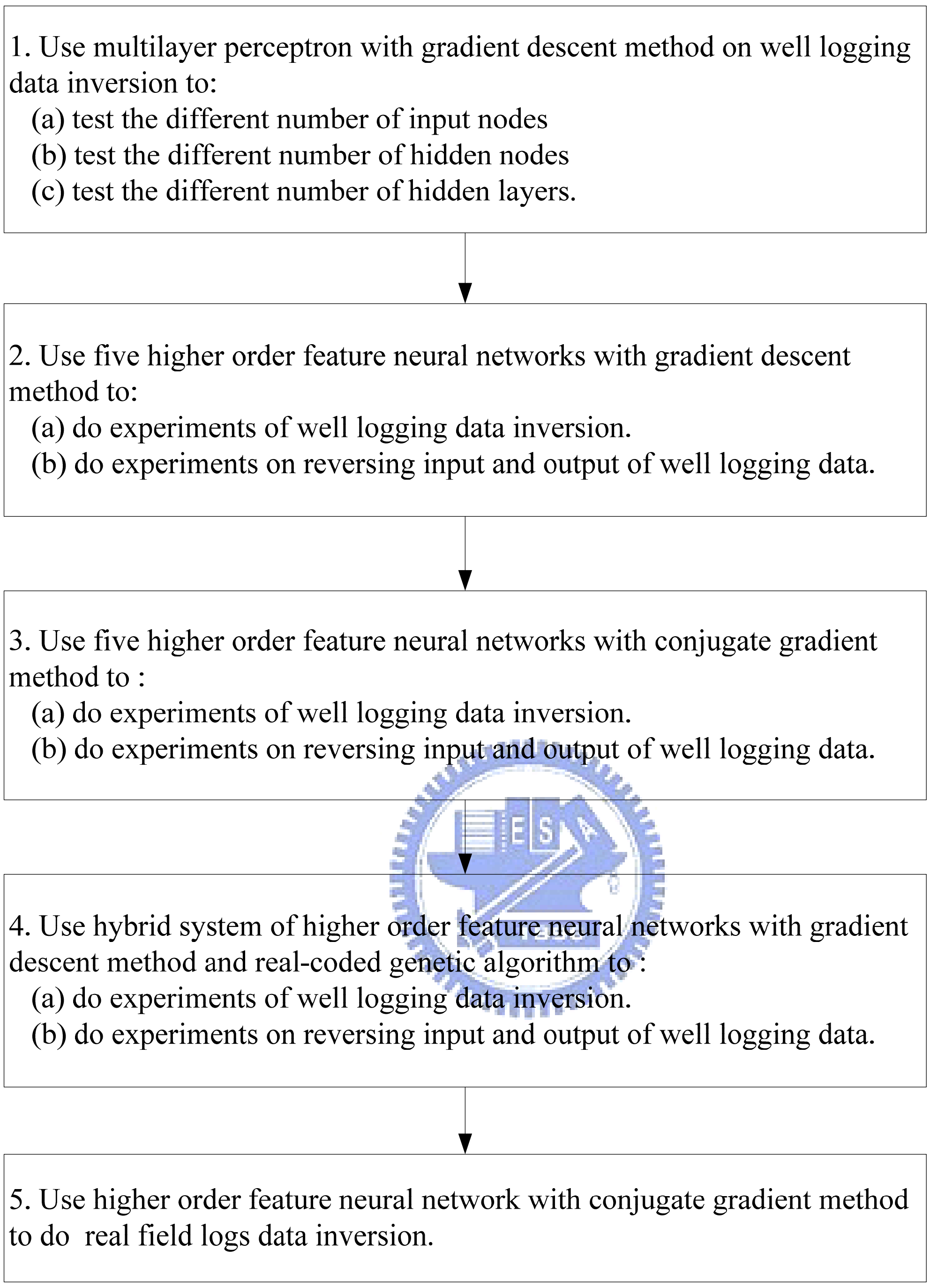
1. Introduction

Multilayer perceptron (MLP) network consists of one or more hidden layers between the input layer and the output layer. Since the back-propagation (BP) learning algorithm that was developed by Rumelhart, Hinton, and Williams in 1986, MLP network has been successfully used on a wide range of applications [1]. The BP learning algorithm uses gradient descent method that calculates the negative gradient of the error with respect to the weights for a given input. The gradient descent method has some drawbacks [2]. For example, it takes long time in training. To speed up the training process, some optimization algorithms, such as conjugate gradient method, are employed to train MLP network. Martin et al. [3] developed a method that utilizes neural network to process the well logging data. Their results showed that the network trained by conjugate gradient was more effective in the well logging inversion than the other existing methods. Their new approach first train the network with conjugate gradient method and later switch to the LM algorithm, to refine the network parameters. In this paper, we use conjugate gradient to train the network, also we expand the features to higher degree.

Another drawback of gradient descent is easy to get trapped at the local minimum. On the other hand, genetic algorithm is a global search algorithm that is inspired by evolution. An implementation of genetic algorithm starts from an initial population that consists of chromosomes. A chromosome is called an individual that is a candidate solution to the target problem. Fitness value of individual is computed through fitness function during evolution. The individual with higher fitness value has more chance to survive and generates offspring in next generation and finally converges to a global minimum. However, since genetic algorithm is usually slow, some researches and applications that combined genetic algorithms and other algorithms were studied. Tan [4] combined genetic algorithm and evolutionary programming which always applies the mutation operator on bias and weights in iteration to train a neural network, and applied on the cancer classification problem. Kinnebrock [5] designed a hybrid method that combines genetic algorithm and neural network. The author demonstrated that the iterative number in training neural network could be reduced, if after iteration the weights are changed by a mutation operation as it is done in genetic algorithm. In our study, we adopt a hybrid method

[6] that combines genetic algorithm and neural network to apply on the well logging data inversion problem. The idea is to refine the connection weight using BP learning with gradient descent after every generation.

The remaining of this paper is as follows. Chapter 2 gives an overview of MLP network, the BP learning algorithm with gradient descent. Chapter 3 describes the conjugate gradient method and the BP learning algorithm with conjugate gradient method. Chapter 4 describes the experiments on well logging data inversion. For the network architecture, we test the different learning rate, the number of hidden nodes, and the number of hidden layers. We design 5 higher-order feature neural nets, which expand the input features to higher degree, to apply on the experiments of well logging data inversion. Comparison of these nets that trained by gradient descent and conjugate gradient will be made. Chapter 5 describes the genetic algorithm, the real-coded genetic algorithm, and the used hybrid method that combines the neural network and genetic algorithm on the experiments of well logging data inversion. Chapter 6 and Chapter 7 give the experiments on real field logs and conclusions. Figure 1-1 is the flowchart of the experiments in our study.



1. Use multilayer perceptron with gradient descent method on well logging data inversion to:
   (a) test the different number of input nodes
   (b) test the different number of hidden nodes
   (c) test the different number of hidden layers.

2. Use five higher order feature neural networks with gradient descent method to:
   (a) do experiments of well logging data inversion.
   (b) do experiments on reversing input and output of well logging data.

3. Use five higher order feature neural networks with conjugate gradient method to :
   (a) do experiments of well logging data inversion.
   (b) do experiments on reversing input and output of well logging data.

4. Use hybrid system of higher order feature neural networks with gradient descent method and real-coded genetic algorithm to :
   (a) do experiments of well logging data inversion.
   (b) do experiments on reversing input and output of well logging data.

5. Use higher order feature neural network with conjugate gradient method to do real field logs data inversion.

Fig 1-1. Experiments flowchart.

2. Training Neural Network with Gradient Descent Method

2.1 Multilayer Perceptron

A MLP network consists of a set of nodes that are fully connected and are organized into multiple layers. Typically MLP network has three or more layers of nodes: an input layer that accepts input data, one or more hidden layers, and an output layer. There are bias parameters in input layer and hidden layer that act as extra nodes with a constant output value of 1. For arbitrary node *j*, the weighted sum of inputs is:

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j(d+1)} \tag{2-1}$$

where *d* is the number of input data to the node, x_i is an input, w_{ji} is the connection weight between node *i* and *j*, and $w_{j(d+1)}$ is the connection weight corresponding to the bias parameter. The output of node *j* is $f(net_j)$ where *f* is the activation function. MLP network usually use differentiable sigmoidal activation function for hidden and output nodes [7]. The formula of sigmoidal function is as follows.

$$f(net) = \frac{1}{1+e^{-net}} \tag{2-2}$$

$$f'(net) = f(net)(1-f(net)) \tag{2-3}$$

MLP network is trained by supervised learning. Supervised learning requires a desired response to be trained, and is based on the comparison between the actual output of network and the desired output. The learning process of MLP network has two phases, one is the forward computation and the other is the backward learning [8]. In forward computation, training pattern is presented to the network from the input layer, and transmit to the hidden layers, the outputs of hidden layers are then transmitted to output layer, finally the real output values are computed. In backward learning phase, the error between the real output value and the desired output value is propagated back to the network to update the connection weights. Figure 2-1 is the example of MLP network with one hidden layer.

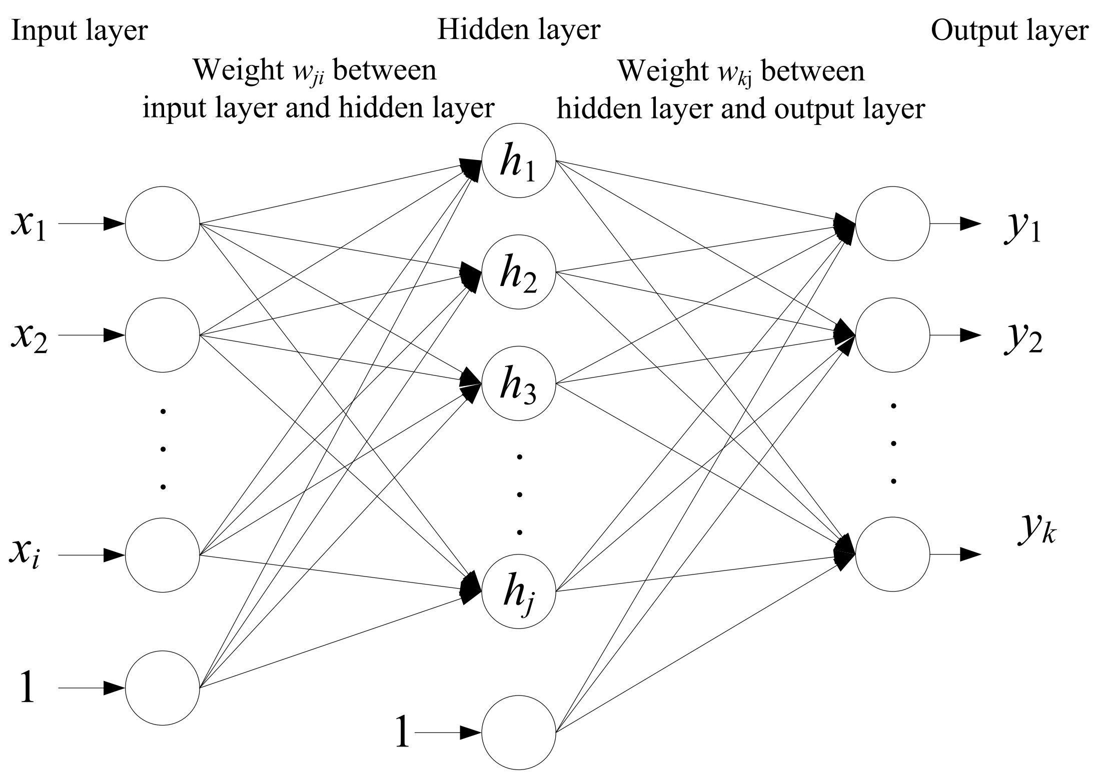

Input layer
Hidden layer
Output layer
Weight w_{ji} between input layer and hidden layer
Weight w_{kj} between hidden layer and output layer
x_1
x_2
x_i
1
h_1
h_2
h_3
h_j
1
y_1
y_2
y_k

Fig. 2-1. MLP network with one hidden layer.

2.2 Gradient Descent Method

Gradient descent method is a function optimization method that uses the negative derivative of the function. MLP network is often trained using the gradient descent method. Assuming we have a MLP network with one hidden layer. There are I nodes in input layer (not including the bias), J nodes in hidden layer (not including the bias), and K nodes in output layer, i, j, and k denote the node index. w_{kj} denotes the weight value connecting output node and hidden node, and w_{ji} denotes the weight value connecting hidden node and input node. The BP algorithm is to modify the weight value iteratively so that the error function of network E is minimized. An error function of a network E is defined as follows.

$$\mathrm{E} = \frac{1}{2}\sum_{k=1}^{K}(d_k - y_k)^2 \tag{2-4}$$

where d_k and y_k denote the desired output value and the real output value of the k^{th} output node respectively.

According to the gradient descent method, the weight change is in the reverse direction of the gradient. The formula of weight change in hidden node and output node is:

$$\Delta w_{kj} = w_{kj}(t+1) - w_{kj}(t) = -\eta \frac{\partial E}{\partial w_{kj}} \tag{2–5}$$

where η is the learning rate and t is the iteration index.

By applying the chain rule on equation (2-5), the weight change in hidden node and output node is:

$$\Delta w_{kj} = \eta(d_k - y_k) f'(net_k) y_j \tag{2-6}$$

Similarly, the weight change in input node and hidden node is:

$$\Delta w_{ji} = w_{ji}(t+1) - w_{ji}(t) = -\eta \frac{\partial E}{\partial w_{ji}} \tag{2-7}$$

By applying the chain rule on equation (2-7), the weight change in input node and hidden node is:

$$\Delta w_{ji} = \eta[\sum_{k=1}^{K}(d_k - y_k) f'(net_k) w_{kj}] f'(net_j) y_i \tag{2-8}$$

2.3 Momentum Term

In equation (2-5) and (2-7), the learning rate determines the speed of convergence by acting as a step size. If it is too large, the network training may fail to convergence because of oscillation. On the other hand, smaller learning rate may get slow convergence in training. To get the advantage of using large learning rate and to lessen the oscillation, Rumelhart et al. (1986) [9] suggested adding a momentum term to stabilize the weight change. The weight update at a given iteration index *t* becomes:

$\Delta w_{kj}(t) = \eta(d_k - y_k) f'(net_k) y_j + \beta \Delta w_{kj}(t-1)$ between hidden and output node, and

$$\Delta w_{ji}(t) = \eta[\sum_{k=1}^{K}(d_k - y_k) f'(net_k) w_{kj}] f'(net_j) y_i + \beta \Delta w_{ji}(t-1)$$

between input and hidden node. β is the momentum parameter.

3. Training Neural Network with Conjugate Gradient Method

3.1 Conjugate Gradient Method

The gradient descent method simply uses the first order (gradient) information, however, using the second order information is often more effective in convergence [10]. There has been technique about the usage of second order information, such as Newton's method [2]. The Newton's method requires storage and expensive computation cost of the inverse Hessian matrix. On the other hand, the conjugate gradient method takes advantages of the second order information but not requires the process of the Hessian matrix [11]. The major difference between the gradient descent method and the conjugate gradient method is in finding the search direction. In gradient descent method, the search direction is always the negative of gradient; however, in conjugate gradient method the search direction is the conjugate direction to improve the search efficiency [12]. 本研究使用 Matlab v7.1 的類神經網路 Toolbox，以共軛梯度法來訓練類神經網路，並應用於井測資料的反推。有關於共軛梯度演算法以及一維線性搜尋法(黃金分割法)，將分別於後面的章節中描述。

共軛梯度法的步驟，是在某一搜尋點 $\mathbf{x}(i)$處選擇一個新的搜尋方向 $\mathbf{d}(i)$，使其與前次的搜尋方向 $\mathbf{d}(i\text{-}1)$滿足 **A**-共軛 [2]，即：

$$\mathbf{d}(i)^T\mathbf{A}\mathbf{d}(i-1) = 0 \qquad (3\text{-}1)$$

然後從搜尋點 $\mathbf{x}(i)$處出發，沿著搜尋方向 $\mathbf{d}(i)$，求得函數 $f(\mathbf{x})$的極小點所需的步長 $\alpha(i)$，亦即求得 $\alpha(i)$使得 $f(\mathbf{x}(i) + \alpha(i)\mathbf{d}(i))$具有最小值。經由所找到的搜尋方向 $\mathbf{d}(i)$與步長 $\alpha(i)$，可以依照疊代式算法計算出下一搜尋點 $\mathbf{x}(i+1)$，即由公式(3-2)以及圖(3-1)所示：

$$\mathbf{x}(i+1) = \mathbf{x}(i) + \alpha(i)\mathbf{d}(i) \qquad (3\text{-}2)$$

圖(3-1)顯示由起始點 $\mathbf{x}(0)$出發，沿著所選定的方向 $\mathbf{d}(0)$移動 $\alpha(0)$距離後，可以計算出新的搜尋點 $\mathbf{x}(1)$，再由點 $\mathbf{x}(1)$開始，決定所需的方向 $\mathbf{d}(1)$與步長 $\alpha(1)$，計算新的搜尋點，依此計算方式一步步逼近問題解。

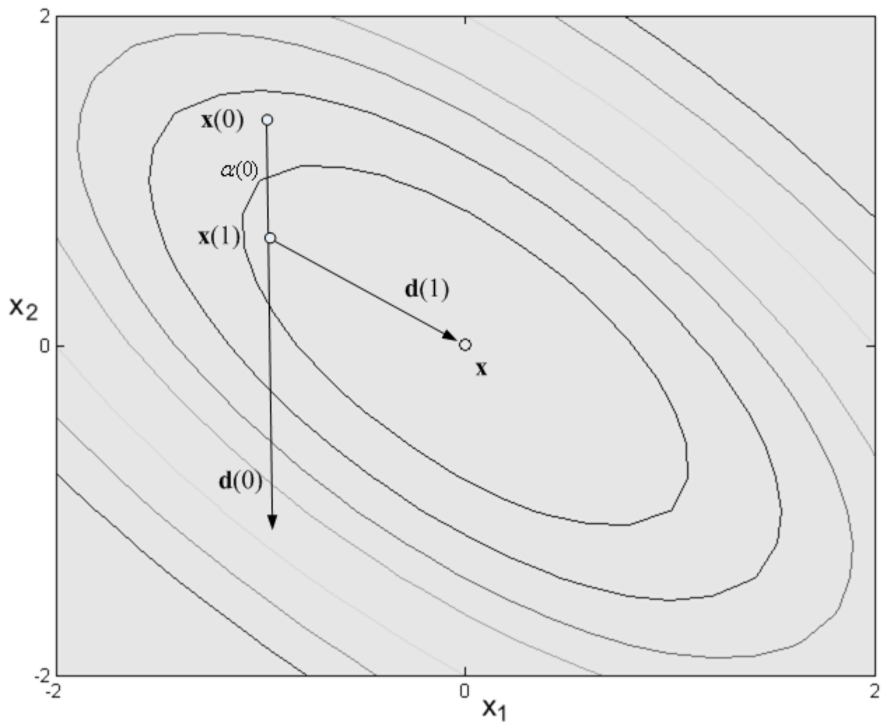


Fig. 3-1. The iterative computation.

接下來的章節，將針對二次函數探討共軛梯度法的性質，3.1.1 節將說明共軛方向，對於求函數極小點問題時，比使用負梯度方向有較好的效率；3.1.2 節說明共軛梯度法的二次終結性(Quadratic Termination)，也就是對於 *n* 維度的二次函數，在 *n* 次 iteration 內就能到達極小點；在探討了共軛方向的性質後，3.1.3 節將說明在共軛梯度法中，求得共軛方向的方法。

3.1.1 Conjugate Directions

本小節說明在二次函數的求極值問題上，在任一搜尋點決定下一個要搜尋的方向時，如果以共軛方向當成搜尋方向，會比梯度下降法使用負梯度的搜尋方向更有效率。

對於一個二次函式有以下的一般表示式 [2]：

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x} + c \qquad (3\text{-}3)$$

其中 **A** 爲對稱正定矩陣，而梯度 [2] 則爲：

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b} \qquad (3\text{-}4)$$

方程組 $\mathbf{Ax}+\mathbf{b}=0$ 的解則爲此二次函數的極小點。圖(3-2)所示爲二次函數 $f(\mathbf{x})=\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Ax}+\mathbf{b}^{\mathrm{T}}\mathbf{x}+c$ 的 3D 曲線圖，其中 $\mathbf{A}=\begin{bmatrix}4 & 2\\ 2 & 8\end{bmatrix}$，$\mathbf{b}=\begin{bmatrix}2\\ 0\end{bmatrix}$，以及 $c=0$，其最小點的位置位於點(-0.57, 0.14)處。

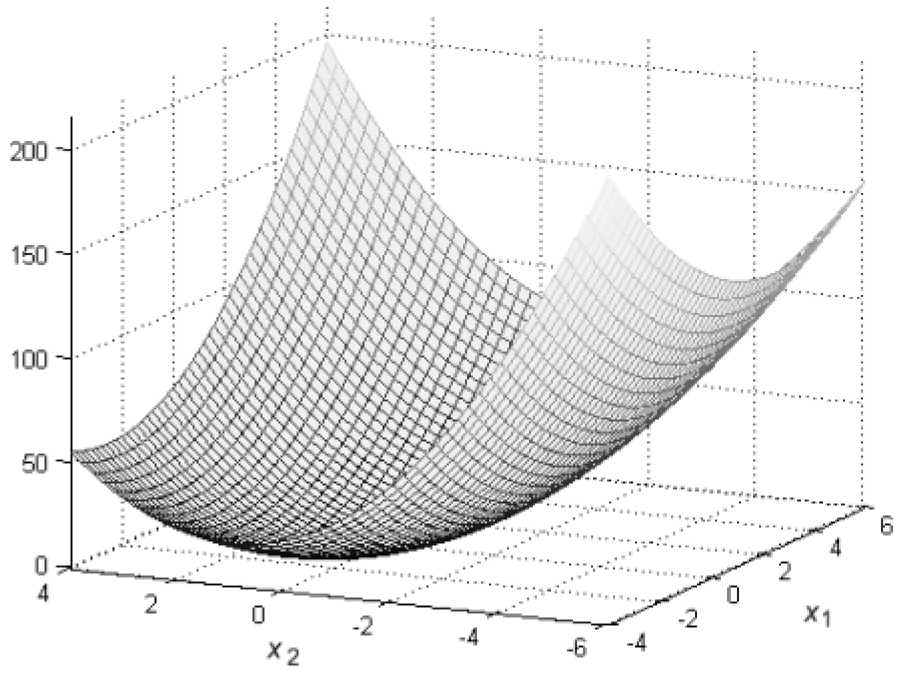


Fig. 3-2. Graph of a quadratic form.

以下的敘述開始說明共軛方向的特性(Jonathan Richard Shewchuk，[12])，爲了方便說明, 以符號 $\mathbf{r}(i)$ 表示在第 i 步時，$\mathbf{Ax}$ 與 $\mathbf{Ax}(i)$ 之間的差異：

$$\mathbf{r}(i)=\mathbf{Ax}-\mathbf{Ax}(i)$$

$$\mathbf{r}(i)=-\mathbf{b}-\mathbf{Ax}(i)$$

$$\mathbf{r}(i)=-\bigtriangledown f(\mathbf{x}(i)) \tag{3-5}$$

另外，以誤差向量 $\Delta\mathbf{x}(i)$，表示真實解與估計值之間的差異：

$$\Delta\mathbf{x}(i)=\mathbf{x}-\mathbf{x}(i) \tag{3-6}$$

關於誤差向量 $\Delta\mathbf{x}$ 與方向 $\mathbf{d}$ 的關係式，可經由公式(3-2)與(3-6)得出：

$$\mathbf{x}(i+1)=\mathbf{x}(i)+\alpha(i)\mathbf{d}(i)$$

$$\mathbf{x}-\mathbf{x}(i+1)=\mathrm{x}-\mathbf{x}(i)-\alpha(i)\mathbf{d}(i)$$

$$\Delta\mathbf{x}(i+1)=\Delta\mathbf{x}(i)-\alpha(i)\mathbf{d}(i) \tag{3-7}$$

爲了搜尋從點 $\mathbf{x}(i)$，沿著方向 $\mathbf{d}(i)$移動的步長 $\alpha(i)$，計算方法是取 $\alpha(i)$使函

數 $f(\mathbf{x}(i)+\alpha(i)\mathbf{d}(i))$ 能夠極小化，可以令 $f(\mathbf{x}(i)+\alpha(i)\mathbf{d}(i))$ 對 α(*i*)的導數爲零，也就是：

$$\frac{d}{d\alpha(i)} f(\mathbf{x}(i)+\alpha(i)\mathbf{d}(i)) = 0$$

$$\frac{d}{d\alpha(i)} f(\mathbf{x}(i+1)) = 0$$

$$f'(\mathbf{x}(i+1))^{\mathrm{T}} \frac{d}{d\alpha(i)} \mathbf{x}(i+1) \ = 0$$

$$-\mathbf{r}^{\mathrm{T}}(i+1)\mathbf{d}(i) = 0$$

$$(\mathbf{Ax}(i+1)–\mathbf{Ax})^{\mathrm{T}}\mathbf{d}(i) = 0 \qquad \text{(by equation (3-5))}$$

$$\mathbf{A}(\mathbf{x}(i+1)–\mathbf{x})^{\mathrm{T}}\mathbf{d}(i) = 0$$

$$–\mathbf{A}\Delta\mathbf{x}(i+1)^{\mathrm{T}}\mathbf{d}(i) = 0$$

$$\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(i+1) = 0 \qquad (3\text{-}8)$$

根據公式(3-1)關於 **A**-共軛的定義，得出 **d**(*i*)與Δ**x**(*i*+1)是兩個滿足 **A**-共軛的方向；繼續整理公式(3-8)以求得步長 α(*i*)：

$$\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}(\Delta\mathbf{x}(i) – \alpha(i)\mathbf{d}(i)) = 0 \qquad \text{(by equation (3-7))}$$

$$\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(i) – \mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\alpha(i)\mathbf{d}(i) = 0$$

$$\alpha(i) = \frac{\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(i)}{\mathbf{d}(i)^{\mathrm{T}}\mathbf{Ad}(i)}$$

$$= -\frac{\mathbf{d}(i)^{\mathrm{T}}\mathbf{r}(i)}{\mathbf{d}(i)^{\mathrm{T}}\mathbf{Ad}(i)}$$

$$\alpha(i) = \frac{-\mathbf{d}(i)^{\mathrm{T}}\nabla f(\mathbf{x}(i))}{\mathbf{d}(i)^{\mathrm{T}}\mathbf{Ad}(i)} \qquad (3\text{-}9)$$

此處的方向 **d**(*i*)若是以 **r**(*i*)來取代，則公式(3-9)即相當於梯度下降法中計算步長的公式，亦即：

$$\alpha(i) = \frac{\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(i)}{\mathbf{r}(i)^{\mathrm{T}}\mathbf{Ar}(i)}$$

$$\alpha(i) = \frac{\nabla f(\mathbf{x}(i))^{\mathrm{T}}\nabla f(\mathbf{x}(i))}{\nabla f(\mathbf{x}(i))^{\mathrm{T}}\mathbf{A}\nabla f(\mathbf{x}(i))}, \text{ for gradient descent method.} \qquad (3\text{-}10)$$

如圖(3-3)與公式(3-8)所示，從起始點 **x**(0)沿著方向 **d**(0)移動，找到最佳步長 α(0)以計算 **x**(1)，必定滿足Δ**x**(1)與 **d**(0)爲 **A**-共軛的條件；也就是說，共軛

梯度法選擇Δ**x**(1)當成是下一個方向 **d**(1)，那麼在求解二維的二次方程式問題時， 兩步之內即可找到極小點，比起使用負梯度的搜尋方向(**r**(1))更有效率。

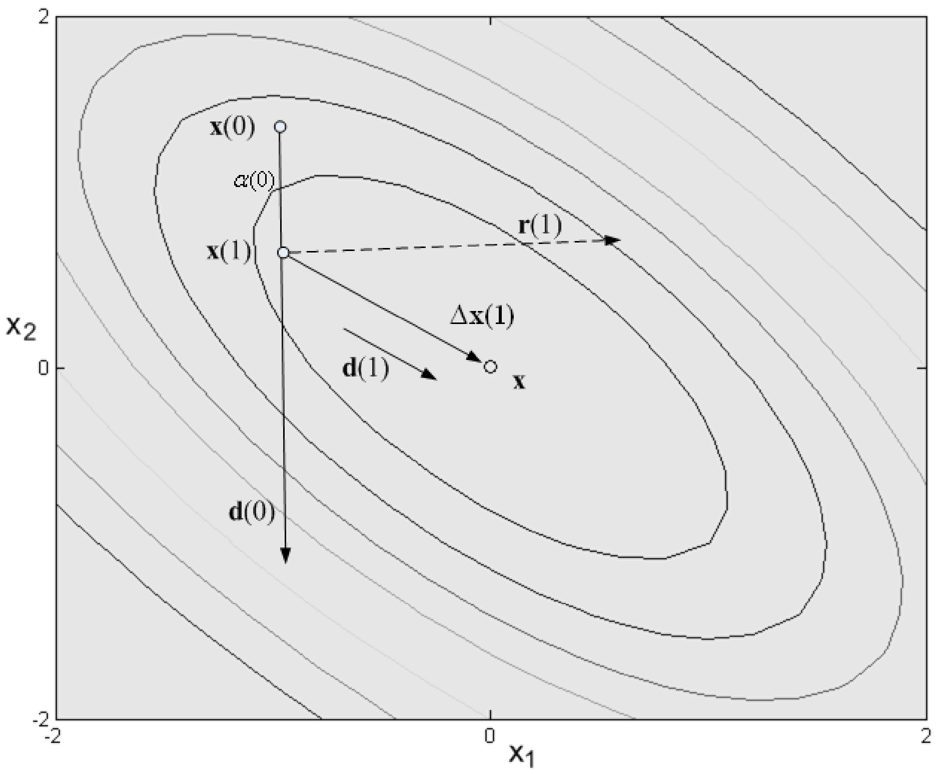


Fig. 3-3. Conjugate gradient method finds minimum in 2 iterations.

3.1.2 Quadratic Termination of Conjugate Gradient Method

上一小節說明，在一個二維的二次函數中，共軛梯度法僅需兩步即能到達極小點，本小節則進一步說明共軛梯度法的二次終結性，亦即在一個 n 維度的二次函數中，在最多 n 次疊代之後就能找到極小點 [2]。也就是經過 n 次疊代之後:

$$\Delta \mathbf{x}(n) = 0$$

以下的敘述開始說明共軛梯度法的二次終結性(Jonathan Richard Shewchuk，[12])，如圖(3-4)所示，起始的誤差向量(虛線部份)Δ**x**(0)是由兩個滿足 **A**-共軛的向量 **d**(0)與Δ**x**(1)線性組合而成，對 n 維的二次函數而言，由於搜尋方向集合{**d**(i)}滿足 **A**-共軛，因此誤差向量Δ**x**(0)可以表示成搜尋方向 **d**(i)

的線性組合：

$$\Delta \mathbf{x}(0) = \sum_{i=0}^{n-1} \delta(i)\, \mathbf{d}(i) \qquad (3\text{-}11)$$

以 $\delta(i)$ 代表集合中任一搜尋方向 $\mathbf{d}(i)$的係數。

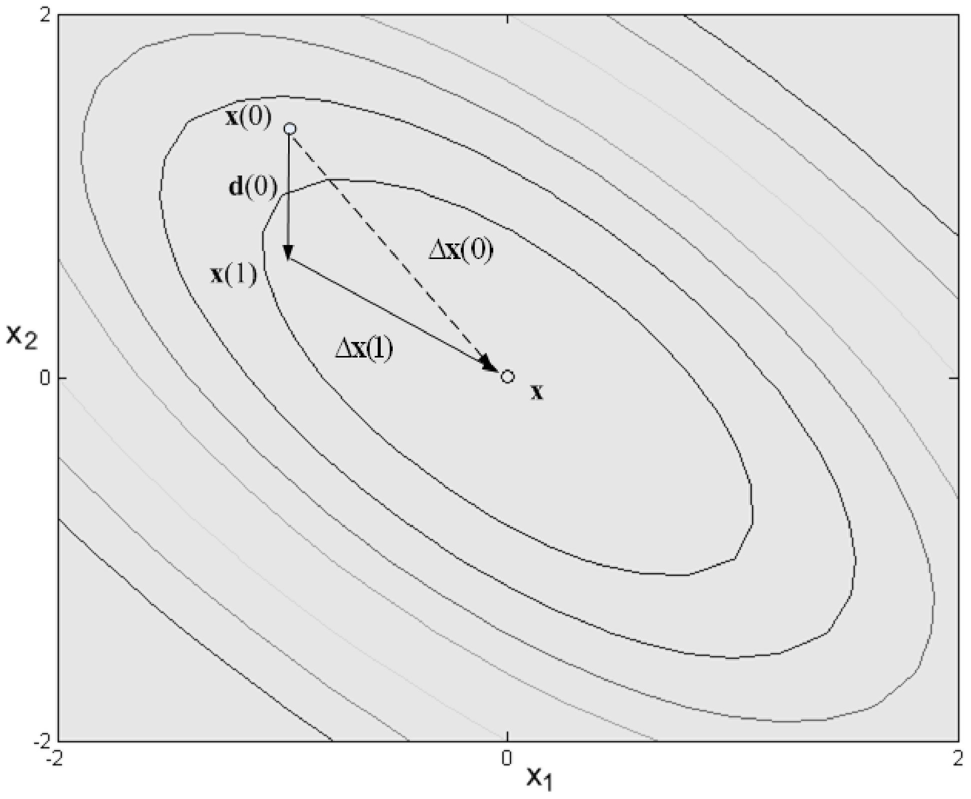


Fig. 3-4. Conjugate gradient method converges in *n* iterations.

爲了計算出 $\delta(i)$ 的值，在此先利用公式(3-7)得出 $\Delta\mathbf{x}(i)$的關係式，也就是：

$\Delta\mathbf{x}(i+1) = \Delta\mathbf{x}(i) - \alpha(i)\mathbf{d}(i)$ (公式(3-7))

將此公式中 i 的數值由 0 代到 i, 得出以下方程組:

$\Delta\mathbf{x}(1) = \Delta\mathbf{x}(0) - \alpha(0)\mathbf{d}(0)$

$\Delta\mathbf{x}(2) = \Delta\mathbf{x}(1) - \alpha(1)\mathbf{d}(1)$

$\Delta\mathbf{x}(3) = \Delta\mathbf{x}(2) - \alpha(2)\mathbf{d}(2)$

…

$\Delta\mathbf{x}(i) = \Delta\mathbf{x}(i-1) - \alpha(i-1)\mathbf{d}(i-1)$

然後將以上方程組相加, 即可得到 $\Delta\mathbf{x}(i)$的關係式:

$$\Delta\mathbf{x}(i) = \Delta\mathbf{x}(0) - \sum_{j=0}^{i-1} \alpha(j)\,\mathbf{d}(j) \qquad (3\text{-}12)$$

利用公式(3-12)有關Δ$\mathbf{x}(i)$的關係式，即可開始計算$\delta(i)$的值。由於搜尋方向$\mathbf{d}(i)$滿足**A**-共軛的性質，因此可以利用將公式(3-11)兩邊各乘以$\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}$的方法，消除大部分的$\delta(i)$項，最後只會留下一個，其中$k$介於 0 與$n$-1：

$$\Delta\mathbf{x}(0) = \sum_{i=0}^{n-1} \delta(i)\,\mathbf{d}(i) \qquad \text{(公式(3-11))}$$

$$\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(0) = \sum_{i=0}^{n-1} \delta(i)\,\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(i)$$

$$\begin{aligned}\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(0) = {} & \delta(0)\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(0) + \\ & \delta(1)\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(1) + \\ & \ldots \\ & \delta(i)\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(i)\end{aligned}$$

$$\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(0) = \delta(k)\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(k) \qquad \text{(by } \mathbf{A}\text{-conjugate property)}$$

$$\delta(k) = \frac{\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(0)}{\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(k)}$$

$$= \frac{\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\left(\Delta\mathbf{x}(0) - \sum_{j=0}^{k-1}\alpha(j)\mathbf{d}(j)\right)}{\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(k)} \qquad \text{(by } \mathbf{A}\text{-conjugate property of } \sum_{j=0}^{k-1}\alpha(j)\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)=0)$$

$$\delta(k) = \frac{\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(k)}{\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(k)} \qquad \left(\text{by } \Delta\mathbf{x}(k) = \Delta\mathbf{x}(0) - \sum_{j=0}^{k-1}\alpha(j)\,\mathbf{d}(j)\right) \qquad (3\text{-}13)$$

由公式(3-9)與(3-13)，可以整理出$\alpha(i)$與$\delta(i)$的關係如下：

$$\begin{aligned}\alpha(i) &= -\frac{\mathbf{d}(i)^{\mathrm{T}}\nabla f(\mathbf{x}(i))}{\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(i)} \\ &= \frac{\mathbf{d}(i)^{\mathrm{T}}\mathbf{r}(i)}{\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(i)} \\ &= \frac{\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(i)}{\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(i)} \\ &= \delta(i)\end{aligned}$$

$$\alpha(i) = \delta(i) \qquad (3\text{-}14)$$

現在可以將公式(3-12)關於Δ$\mathbf{x}(i)$的關係式做以下的改寫：

$$\Delta\mathbf{x}(i) = \Delta\mathbf{x}(0) - \sum_{j=0}^{i-1}\alpha(j)\,\mathbf{d}(j)$$

$$= \sum_{j=0}^{n-1}\delta(j)\,\mathbf{d}(j) - \sum_{j=0}^{i-1}\delta(j)\mathbf{d}(j)$$

$$\Delta\mathbf{x}(i) = \sum_{j=\mathrm{i}}^{n-1}\delta(j)\mathbf{d}(j) \qquad (3\text{-}15)$$

上式的意義表明，在第 i step 時的誤差向量Δ$\mathbf{x}(i)$，等於從第 i 到第 n-1 step 之間的項目總和，也就是說每經過一次疊代，就會有一個項目被消除掉，因此，經過 n 次疊代之後，Δ$\mathbf{x}(n) = 0$，由此得知，在 n 維的二次函數中，最多經過 n 次的疊代即能到達極小點。

以下為 $n = 3$ 的實例說明：

$$\Delta\mathbf{x}(0) = \delta(0)\mathbf{d}(0) + \delta(1)\mathbf{d}(1) + \delta(2)\mathbf{d}(2)$$

$$\Delta\mathbf{x}(1) = \delta(1)\mathbf{d}(1) + \delta(2)\mathbf{d}(2)$$

$$\Delta\mathbf{x}(2) = \delta(2)\mathbf{d}(2)$$

$$\Delta\mathbf{x}(3) = 0$$

每一次 iteration 之後，都會減少一項δ 與 **d** 的乘積項，經過 3 次 iteration 後，得到誤差向量 $\Delta\mathbf{x}(3) = 0$ ，也就是已到達極小點。

3.1.3 Construction of Conjugate Directions

由 3.1.1 與 3.1.2 小節可以得知，採用共軛方向當成每一次疊代的搜尋方向，能夠有比較好的搜尋效能。因此，本小節即介紹求得一組共軛方向 **d**(0)，**d**(1)，…，**d**(n-1)的方法 (Jonathan Richard Shewchuk，[12])，其中 n 代表二次函數的維度。

求得共軛方向的過程稱為Gram-Schmidt Conjugate Process [12]。 假設空間中有n 個獨立的向量**u**(0)，**u**(1)，…，**u**(n-1)，要建立i>0 的任一方向**d**(i)，可以取**u**(i)減去它的其中一個未滿足與前方向**A**-共軛的分量。如圖(3-5)所示，從兩個獨立的向量**u**(0)與**u**(1)開始，令**d**(0) = **u**(0)，而**u**(1)是由兩個分量**u***與$\mathbf{u}^+$組合而成，其中**u***與前一次方向**d**(0)滿足**A**-共軛，而$\mathbf{u}^+$則是與**d**(0)平行的分量，也就是**u**(1)在**d**(0)的投影，用**u**(1)減去$\mathbf{u}^+$之後，留下來的分量**u***就是新的共軛方向

d(1)，在Gram-Schmidt正交化過程中, 投影可以用內積乘上**d**(0)來表示，在此則以係數$\beta(1,0)$表示：

$$\mathbf{u}(1) = \mathbf{u}^{+} + \mathbf{u}^{*}$$
$$\mathbf{u}^{*} = \mathbf{u}(1) - \mathbf{u}^{+}$$
$$\mathbf{d}(1) = \mathbf{u}(1) - \mathbf{u}^{+} = \mathbf{u}(1) + \beta(1,0)\,\mathbf{d}(0)$$

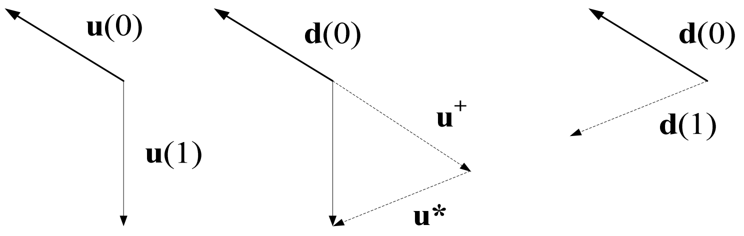


Fig 3-5. Gram-Schmidt conjugate process.

同理，**d**(2)等於 **u**(2)減去在 **d**(0)的投影以及在 **d**(1)的投影，也就是：

$$\mathbf{d}(2) = \mathbf{u}(2) + \beta(2,0)\,\mathbf{d}(0) + \beta(2,1)\,\mathbf{d}(1)$$

因此，對於任意搜尋方向 **d**(*i*)可以表示成：

$$\mathbf{d}(i) = \mathbf{u}(i) + \sum_{k=0}^{i-1} \beta(i,k)\mathbf{d}(k), \qquad i > k \tag{3-16}$$

為了計算$\beta(i,k)$的值，將上述公式(3-16)乘上$\mathbf{d}(j)^{\mathrm{T}}\mathbf{A}$，其中$i > j$：

$$\mathbf{d}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) = \mathbf{u}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) + \sum_{k=0}^{i-1} \beta(i,k)\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)$$

$$0 = \mathbf{u}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) + \beta(i,0)\mathbf{d}(0)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) + \beta(i,1)\mathbf{d}(1)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) + \ldots + \beta(i,k)\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)$$

$$0 = \mathbf{u}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) + \beta(i,j)\mathbf{d}(j)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j), \qquad i > j \quad \text{(by } \mathbf{A}\text{-conjugate property)}$$

$$\beta(i,j) = -\frac{\mathbf{u}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)}{\mathbf{d}(j)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)}, \qquad i > j \tag{3-17}$$

如公式(3-16)所示，Gram-Schmidt Conjugate Process 的缺點在於，在建立一個新的搜尋方向 **d**(*i*)時，之前的 *i*-1 次方向都必須保留下來，一個改善的方法是，使用 **r**(*i*)來取代之前使用的 **u**(*i*) [12]，也就是說，令 **u**(*i*) = **r**(*i*)，因此，計算搜尋方向的公式(3-16)可以改寫成公式(3-18)的形式，公式(3-17)也可以改寫成公

式(3-19)。

$$\mathbf{d}(i)=\mathbf{r}(i)+\sum_{k=0}^{i-1}\beta(i,k)\mathbf{d}(k),\qquad i>k \tag{3-18}$$

$$\beta(i,j)=-\frac{\mathbf{r}(i)^{\mathrm{T}}\mathbf{Ad}(j)}{\mathbf{d}(j)^{\mathrm{T}}\mathbf{Ad}(j)},\qquad i>j \tag{3-19}$$

爲了求出$\mathbf{d}(k)$與$\mathbf{r}(i)$的關係(其中 $i>k$)，將公式(3-15)乘上$\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}$：

$$\Delta\mathbf{x}(i)=\sum_{j=\mathrm{i}}^{n-1}\delta(j)\mathbf{d}(j)\qquad\text{(公式(3-15))}$$

$$\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(i)=\sum_{j=i}^{n-1}\delta(j)\mathbf{d}(j)^{\mathrm{T}}\mathbf{Ad}(k)$$

$$\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}\Delta\mathbf{x}(i)=0$$

$$\mathbf{d}(k)^{\mathrm{T}}\mathbf{A}(\mathbf{x}-\mathbf{x}(i))=0$$

$$\mathbf{d}(k)^{\mathrm{T}}(\mathbf{Ax}-\mathbf{Ax}(i))=0$$

$$\mathbf{d}(k)^{\mathrm{T}}\mathbf{r}(i)\quad=0,\qquad i>k\qquad\text{(by }\mathbf{A}\text{-conjugate property)} \tag{3-20}$$

由上式(3-20)得知，$\mathbf{r}(i)$與舊的搜尋方向 $\mathbf{d}(k)$之間成正交的關係。

接下來說明，在任一點 $\mathbf{x}(i)$上建立新搜尋方向 $\mathbf{d}(i)$的方法，在過程中也會說明在使用 $\mathbf{r}(i)$取代 $\mathbf{u}(i)$的情況下，則可以不用保留所有舊方向，僅利用前一次的方向 $\mathbf{d}(i\text{-}1)$，即可建立新方向 $\mathbf{d}(i)$。

首先，利用公式(3-18)，在式子的兩邊分別取與項目 $\mathbf{r}(j)$的內積，其中 $j\neq i$：

$$\mathbf{d}(i)=\mathbf{r}(i)+\sum_{k=0}^{i-1}\beta(i,k)\mathbf{d}(k),\qquad i>k\qquad\text{(公式(3-18))}$$

$$\mathbf{d}(i)^{\mathrm{T}}\mathbf{r}(j)=\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(j)+\sum_{k=0}^{i-1}\beta(i,k)\mathbf{d}(k)^{\mathrm{T}}\mathbf{r}(j)$$

$$0\;=\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(j)\qquad\text{(by equation }\mathbf{d}(i)^{\mathrm{T}}\mathbf{r}(j)=0) \tag{3-21}$$

另外可得：

$$\mathbf{d}(i)^{\mathrm{T}}\mathbf{r}(i)=\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(i) \tag{3-22}$$

另外，由公式(3-20)的推導過程中，可以得出 $\mathbf{r}(i)$與$\Delta\mathbf{x}(i)$的關係式，也就是 $\mathbf{r}(i)=\mathbf{A}\Delta\mathbf{x}(i)$，因此：

$$\mathbf{r}(i+1) = \mathbf{A}\,\Delta\,\mathbf{x}(i+1)$$

$$\mathbf{r}(i+1) = \mathbf{A}(\Delta\,\mathbf{x}(i) - \alpha(i)\mathbf{d}(i)) \qquad \text{(由公式(3-7))}$$

$$\mathbf{r}(i+1) = \mathbf{A}\,\Delta\,\mathbf{x}(i) - \mathbf{A}\alpha(i)\mathbf{d}(i)$$

$$\mathbf{r}(i+1) = \mathbf{r}(i) - \alpha(i)\mathbf{A}\mathbf{d}(i) \tag{3-23}$$

接下來，爲了求出$\beta(i,j)$的值，將上述公式(3-23)以 j 來取代 i，並且與項目 $\mathbf{r}(i)$做內積，這樣做的原因是要求得$\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)$，以便應用於公式(3-19)中求得$\beta(i,j)$的值：

$$\mathbf{r}(j+1) = \mathbf{r}(j) - \alpha(j)\mathbf{A}\mathbf{d}(j)$$

$$\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(j+1) = \mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(j) - \alpha(j)\,\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)$$

$$\alpha(j)\,\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) = \mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(j) - \mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(j+1)$$

$$\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) = \frac{1}{\alpha(j)}(\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(j) - \mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(j+1))$$

從以下三種狀況來考慮$\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)$項:

(a) $\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) = \dfrac{1}{\alpha(i)}\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(i), \qquad i=j$

(b) $\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) = -\dfrac{1}{\alpha(i\text{-}1)}\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(i), \qquad i=j+1$

(c) $\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j) = 0, \qquad i>j+1$

在此忽略上述的狀況(a)，因爲當 $i=j$ 的時候，計算$\beta(i,j)$無意義，因此，依上述狀況(b)與狀況(c)分別將$\beta(i,j)$重新寫成：

(1) $$\beta(i,j) = -\frac{\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)}{\mathbf{d}(j)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)} = \frac{1}{\alpha(i-1)}\frac{\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(i)}{\mathbf{d}(i\text{-}1)^{\mathrm{T}}\mathbf{A}\mathbf{d}(i\text{-}1)}, \quad i=j+1 \tag{3-24}$$

(2) $$\beta(i,j) = -\frac{\mathbf{r}(i)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)}{\mathbf{d}(j)^{\mathrm{T}}\mathbf{A}\mathbf{d}(j)} = 0, \qquad i>j+1 \tag{3-25}$$

由於$\mathbf{d}(i) = \mathbf{r}(i) + \sum_{j=0}^{i-1}\beta(i,j)\mathbf{d}(j)$，並且如公式(3-25)所示，$\beta(i,j) = 0$ if $i > j+1$，因此得知，欲求得方向 $\mathbf{d}(i)$，不再需要第 i-1 步之前的所有舊方向，只要知道$\beta(i,j)$的值(其中 j = i-1)，即可藉由前一次的方向 $\mathbf{d}(i\text{-}1)$求得。由此，改寫

$\beta(i,j)=\beta(i,(i\text{-}1))=\beta(i)$。經過以上的討論，可以整理出當 i>0 時，任一點 $\mathbf{x}(i)$ 的搜尋方向 $\mathbf{d}(i)$的計算公式：

$$\mathbf{d}(i)=\mathbf{r}(i)+\beta(i,j)\mathbf{d}(j)=\mathbf{r}(i)+\beta(i)\mathbf{d}(i\text{-}1), \qquad i>0$$

其中

$$\beta(i)=\frac{\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(i)}{\mathbf{d}(i\text{-}1)^{\mathrm{T}}\mathbf{r}(i\text{-}1)} \qquad (\text{由公式}(3-24)\text{與}(3-9))$$

$$=\frac{\mathbf{r}(i)^{\mathrm{T}}\mathbf{r}(i)}{\mathbf{r}(i\text{-}1)^{\mathrm{T}}\mathbf{r}(i\text{-}1)} \qquad (\text{由 } \mathbf{d}(i\text{-}1)^{\mathrm{T}}\mathbf{r}(i\text{-}1)=\mathbf{r}(i\text{-}1)^{\mathrm{T}}\mathbf{r}(i\text{-}1))$$

由於 $\mathbf{r}(i)=-\nabla f(\mathbf{x}(i))$，因此可以將計算搜尋方向 $\mathbf{d}(i)$的公式整理如下：

$$\mathbf{d}(0)=-\nabla f(\mathbf{x}(0))$$

$$\mathbf{d}(i)=-\nabla f(\mathbf{x}(i))+\beta(i)\mathbf{d}(i\text{-}1)\,, \qquad i>0$$

其中

$$\beta(i)=\frac{\nabla f(\mathbf{x}(i))^{\mathrm{T}}\nabla f(\mathbf{x}(i))}{\nabla f(\mathbf{x}(i\text{-}1))^{\mathrm{T}}\nabla f(\mathbf{x}(i\text{-}1))}$$

以下針對二次函數求極小值的問題，以共軛梯度法求解的過程，寫成演算法的形式：

Algorithm 3-1: To minimize quadratic function using conjugate gradient method.

Input: Start point **x**(0).

Output: The solution **x**.

Step1:

設 iteration index $i=0$

計算搜尋點 **x**(0)處的梯度$\nabla f(\mathbf{x}(0))$，並且將第一次的搜尋方向設為負梯度方向：

$$\mathbf{d}(0)=-\nabla f(\mathbf{x}(0))$$

計算步長 $\alpha(0)=\dfrac{-\mathbf{d}(0)^{\mathrm{T}}\nabla f(\mathbf{x}(0))}{\mathbf{d}(0)^{\mathrm{T}}\mathbf{A}\mathbf{d}(0)}$

更新到搜尋點 $\mathbf{x}(1)=\mathbf{x}(0)+\alpha(0)\mathbf{d}(0)$

Step2:

設 $i = i + 1$

計算搜尋點 $\mathbf{x}(i)$處的梯度$\nabla f(\mathbf{x}(i))$

計算係數 $\beta(i)$：

$$\beta(i) = \frac{\nabla f(\mathbf{x}(i))^{\mathrm{T}} \nabla f(\mathbf{x}(i))}{\nabla f(\mathbf{x}(i\text{-}1))^{\mathrm{T}} \nabla f(\mathbf{x}(i\text{-}1))}$$

計算新的搜尋方向 $\mathbf{d}(i)$：

$$\mathbf{d}(i) = -\nabla f(\mathbf{x}(i)) + \beta(i)\mathbf{d}(i\text{-}1)$$

計算步長 $\alpha(i)$：

$$\alpha(i) = \frac{-\mathbf{d}(i)^{\mathrm{T}} \nabla f(\mathbf{x}(i))}{\mathbf{d}(i)^{\mathrm{T}} \mathbf{A}\mathbf{d}(i)}$$

更新到搜尋點 $\mathbf{x}(i+1) = \mathbf{x}(i) + \alpha(i)\mathbf{d}(i)$

Step3:

計算搜尋點 $\mathbf{x}(i+1)$處的梯度，

如果梯度等於零代表已經找到極小點，則結束演算法，並輸出最佳解

$\mathbf{x} = \mathbf{x}(i+1)$；

否則，回到 step 2.

Example 1

To find the minimum of $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}\mathbf{x}$ using conjugate gradient

The gradient vector $\nabla f(\mathbf{x}) = \begin{bmatrix} 3x_1 + 2x_2 \\ 2x_1 + 3x_2 \end{bmatrix}$

Set starting point $\mathbf{x}(0) = \begin{bmatrix} -0.9 \\ 0.2 \end{bmatrix}$

Iteration 1

設第一次搜尋方向爲負梯度方向：

$$\mathbf{d}(0) = -\nabla f(\mathbf{x}(0)) = \begin{bmatrix} 2.3 \\ 1.2 \end{bmatrix}$$

計算步長，使用公式 $\alpha(0) = \dfrac{-\mathbf{d}(0)^{\mathrm{T}} \nabla f(\mathbf{x}(0))}{\mathbf{d}(0)^{\mathrm{T}} \mathbf{A}\mathbf{d}(0)}$：

$$\alpha(0) = -[2.3\ 1.2]\begin{bmatrix}-2.3\\-1.2\end{bmatrix} \ / \ [2.3\ 1.2]\begin{bmatrix}3 & 2\\2 & 3\end{bmatrix}\begin{bmatrix}2.3\\1.2\end{bmatrix} = 0.2155$$

更新到搜尋點 $\mathbf{x}(1)$：

$$\mathbf{x}(1) = \mathbf{x}(0) + \alpha(0)\mathbf{d}(0) = \begin{bmatrix}-0.9\\0.2\end{bmatrix} + 0.2155\begin{bmatrix}2.3\\1.2\end{bmatrix} = \begin{bmatrix}-0.4044\\0.4586\end{bmatrix}$$

Iteration 2

計算搜尋點 $\mathbf{x}(1)$處的梯度：

$$\nabla f(\mathbf{x}(1)) = \begin{bmatrix}3 & 2\\2 & 3\end{bmatrix}\begin{bmatrix}-0.4044\\0.4586\end{bmatrix} = \begin{bmatrix}-0.2957\\0.5672\end{bmatrix}$$

計算 $\beta(1) = \dfrac{\nabla f(\mathbf{x}(1))^{\mathrm{T}}\nabla f(\mathbf{x}(1))}{\nabla f(\mathbf{x}(0))^{\mathrm{T}}\nabla f(\mathbf{x}(0))}$ ：

$$\beta(1) = [-0.2957\ 0.5672]\begin{bmatrix}-0.2957\\0.5672\end{bmatrix} \ / \ [-2.3\ -1.2]\begin{bmatrix}-2.3\\-1.2\end{bmatrix} = 0.0608$$

計算搜尋方向 $\mathbf{d}(1)$：

$$\mathbf{d}(1) = -\nabla f(\mathbf{x}(1)) + \beta(1)\mathbf{d}(0) = \begin{bmatrix}0.2957\\-0.5672\end{bmatrix} + 0.0608\begin{bmatrix}2.3\\1.2\end{bmatrix}$$

$$= \begin{bmatrix}0.4355\\-0.4942\end{bmatrix}$$

計算步長，使用公式 $\alpha(1) = \dfrac{-\mathbf{d}(1)^{\mathrm{T}}\nabla f(\mathbf{x}(1))}{\mathbf{d}(1)^{\mathrm{T}}\mathbf{A}\mathbf{d}(1)}$ ：

$$\alpha(1) = -[0.4355\ -0.4942]\begin{bmatrix}-0.2957\\0.5672\end{bmatrix} / \ [0.4355\ -0.4942]\begin{bmatrix}3 & 2\\2 & 3\end{bmatrix}\begin{bmatrix}0.4355\\-0.4942\end{bmatrix}$$

$$= 0.9281$$

更新到搜尋點 $\mathbf{x}(2)$：

$$\mathbf{x}(2) = \mathbf{x}(1) + \alpha(1)\mathbf{d}(1) = \begin{bmatrix}-0.4043\\0.4586\end{bmatrix} + 0.9281\begin{bmatrix}0.4355\\-0.4942\end{bmatrix}$$

$$= \begin{bmatrix}-0.0001\\-0.0001\end{bmatrix} \approx \begin{bmatrix}0\\0\end{bmatrix}$$

計算搜尋點 $\mathbf{x}(2)$處的梯度：

$$\nabla f(\mathbf{x}(2)) = \begin{bmatrix}3 & 2\\2 & 3\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix},$$

由於在搜尋點 **x**(2)處的梯度為 0，因此可以得知已經到達極小點的位置。

As expected, the quadratic function can be minimized in 2 iterations as shown in Figure 3-6, because the number of dimension of parameter is 2. The results of iterations are list in Table 3-1. Figure 3-7 and Table 3-2 is the result of same example but using the gradient descent method with exact step size. The gradient descent method needs more iterations than conjugate gradient method to converge.

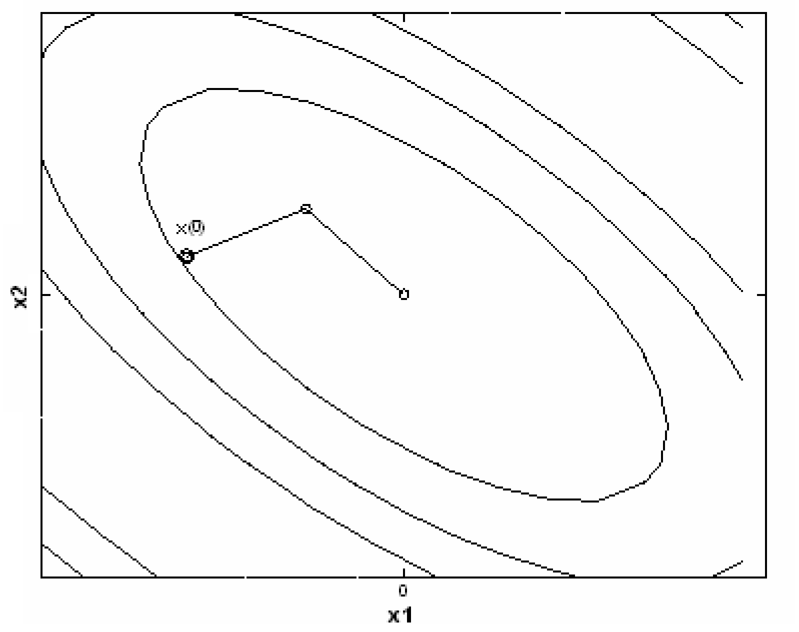


Fig. 3-6. Conjugate gradient method with exact step size.

Table 3-1. Result of conjugate gradient for 2 iterations.

Iteration	1	2
Updated point ($x1$,$x2$)	–0.4044 0.4586	–0.0001 –0.0001
Gradient	–0.2957 0.5672	0 0

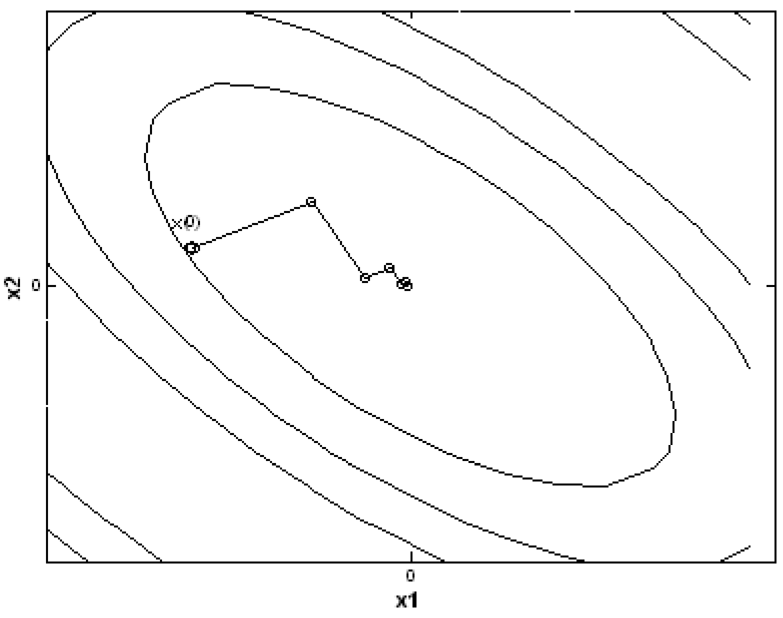


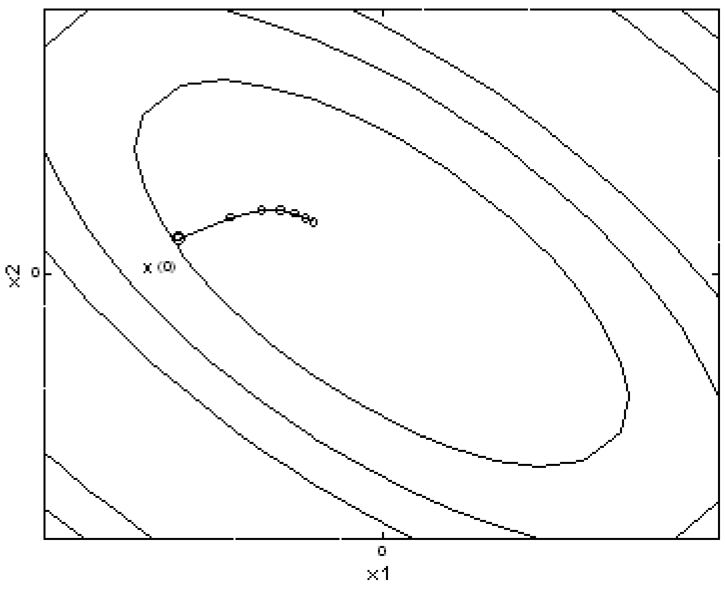
Fig. 3-7. Gradient descent method with exact step size.

Table 3-2. Result of gradient descent for 6 iterations.

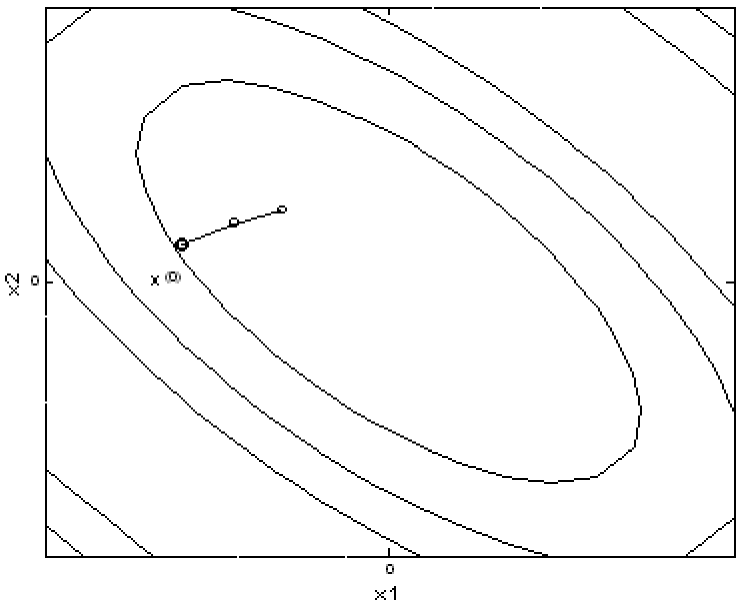
Iteration	1	2	3	4	5	6
Updated point ($x1$,$x2$)	–0.4044 0.4586	–0.1867 0.0415	–0.0839 0.0952	–0.0387 0.0086	–0.0174 0.0197	–0.0080 0.0018
Gradient	–0.2957 0.5672	–0.4772 –0.2490	–0.0614 0.1177	–0.0990 –0.0517	–0.0127 0.0244	–0.0205 –0.0107

Example 2

Consider the same problem in example 1, we now set the step size $\alpha(i)$ a constant. Figure 3-8 shows that if the step size is set too small, the convergence is slow down in gradient descent method. And in conjugate gradient method, it can not find the minimum in 2 iterations. On the other hand, if the step size is set too large, oscillation happens in both gradient descent method and conjugate gradient method as shown in Figure 3-9.

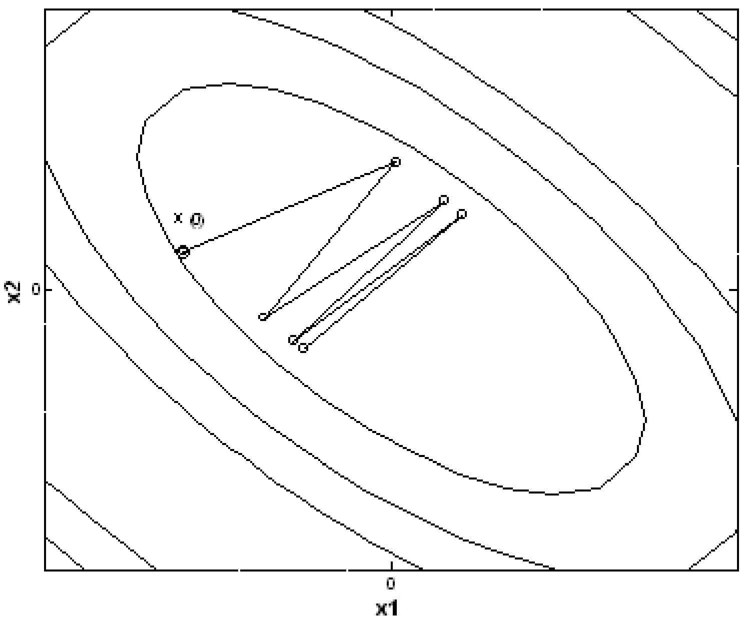


(a) The gradient descent method with step size 0.1.

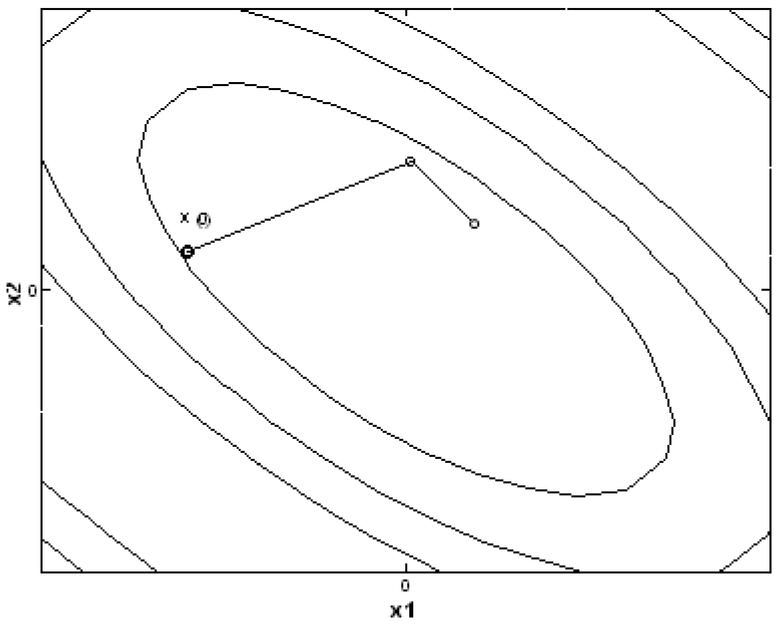


(b) The conjugate gradient method with step size 0.1.

Fig. 3-8. The curve in (a) gradient descent and (b) conjugate gradient when step size is 0.1.



(a) The gradient descent method with step size 0.4.



(b) The conjugate gradient method with step size 0.4.

Fig. 3-9. The curve in (a) gradient descent and (b) conjugate gradient when step size is 0.4.

以上章節關於共軛梯度法的討論，皆是針對二次函數而言，對於將共軛梯度法應用於其他領域，例如類神經網路的訓練上，由於類神經網路的誤差函數並非二次函數，因此在每一個 iteration 的步長 α ，不能直接代公式(3-9)計算得到，可以使用一維線性搜尋(例如黃金分割法 [2])來計算；另外，對於非二次函數的找尋極小值問題，一般皆無法在 *n* 次 iteration(*n* 為問題變數的總數)內收

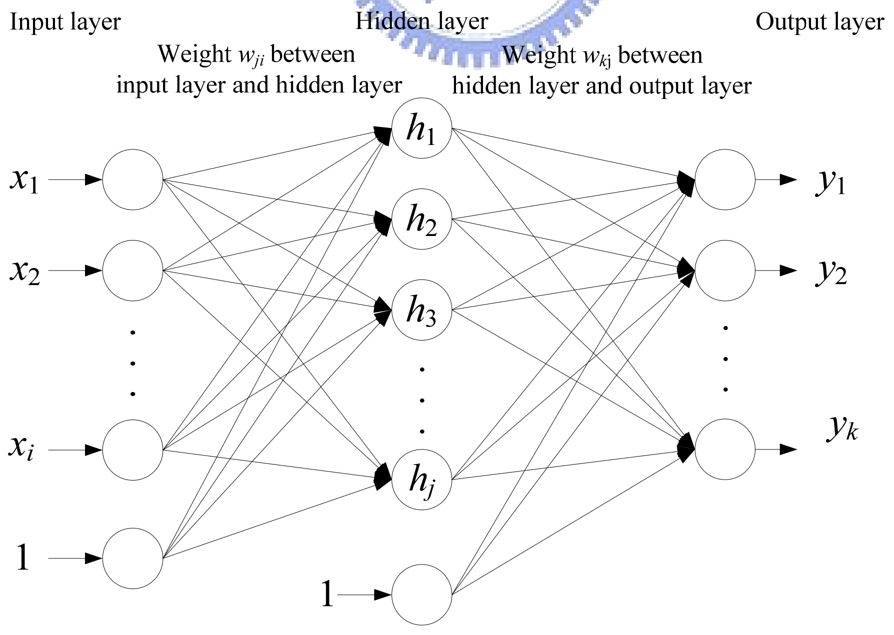
斂，因此在執行 iteration 到達 n 次時，將搜尋方向重新設定爲負梯度方向 (Hagan, *Neural Network Design*, page 12-18, [2])，再繼續按照共軛梯度法的原則繼續執行。

3.2 Back-Propagation with Conjugate Gradient Method

Conjugate gradient method is one of the numerical optimization techniques and can be used to train MLP networks. The algorithm that describes how the conjugate gradient method is used to train MLP networks is called conjugate gradient back-propagation (CGBP). CGBP is a batch mode algorithm (Hagan, *Neural Network Design*, page 12-18, [2]), as the gradient is computed after the entire training set has been presented to the network. 如圖(3-10)所示，一個多層的網路其所有的連結權重值以一個向量來表示，即：

$$\mathbf{w} = [w_{ji}, \ldots , w_{j(i+1)}, w_{kj}, \ldots , w_{k(j+1)}]^{T}$$

其中w_{ji} 是連接隱藏層單元 j 到輸入層單元i 的權重值，w_{kj}是連接輸出層單元k 到隱藏層單元 j 的權重值。i = 1, …, I，j = 1, …, J, and k = 1, …, K，其中I， J，與 K 分別代表輸入層，隱藏層，與輸出層的單元數。



Input layer
Hidden layer
Output layer
Weight w_{ji} between input layer and hidden layer
Weight w_{kj} between hidden layer and output layer
x_1
x_2
x_i
1
h_1
h_2
h_3
h_j
1
y_1
y_2
y_k

Fig. 3-10. MLP network

The training process is to minimize the error function E iteratively, and E is the network error over the entire training set:

$$\mathrm{E} = \frac{1}{2N}\sum_{n=1}^{N}\sum_{k=1}^{K}(d_{nk} - y_{nk})^2 \tag{3-26}$$

N 是訓練樣本的數量，K 是輸出神經元的數量，d_{nk} 與 y_{nk} 分別是使用第 n 個樣本作訓練時，第 k 個輸出神經元的期望輸出值與實際輸出值。其中 $y_{nk} = f(net_{nk})$，net_{nk} 爲該神經元的輸入總和，$f(net) = \dfrac{1}{1+e^{-net}}$ 爲 activation function。梯度則爲網路誤差函數對權重值的導數：

$$\mathbf{g} = \frac{\partial \mathrm{E}}{\partial \mathbf{w}}$$

梯度的計算分爲兩層，分別是輸出層與隱藏層。對於輸出層而言，梯度的計算方式如下：

$$g_{kj} = -\frac{1}{N}\sum_{n=1}^{N}[\,(d_{nk} - y_{nk})f'(net_{nk})y_{nj}\,] \tag{3-27}$$

對於隱藏層而言，則爲：

$$g_{ji} = -\frac{1}{N}\sum_{n=1}^{N}\left\{[\,\sum_{k=1}^{K}(d_{nk} - y_{nk})f'(net_{nk})w_{kj}\,]f'(net_{nj})y_{ni}\right\} \tag{3-28}$$

共軛梯度倒傳遞演算法藉著疊代式運算，以一步一步更新權重值來減少網路誤差值。從任意給定的起始權重值 $\mathbf{w}(0)$ 開始，選取第一個搜尋方向爲 $\mathbf{d}(0)$ 爲負梯度的方向：

$$\mathbf{d}(0) = -\mathbf{g}(0) \tag{3-29}$$

更新權重值是依照以下的公式進行：

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha(t)\mathbf{d}(t) \tag{3-30}$$

$$\mathbf{d}(t+1) = -\mathbf{g}(t+1) + \beta(t+1)\mathbf{d}(t) \tag{3-31}$$

要完成上述更新類神經網路權重值程序，必須要求得係數 α 與 β 的值，其中 β 爲計算搜尋方向所需的係數，可使用以下公式計算：

$$\beta(t) = \frac{\mathbf{g}(t)^{\mathrm{T}}\mathbf{g}(t)}{\mathbf{g}(t\text{-}1)^{\mathrm{T}}\mathbf{g}(t\text{-}1)} \tag{3-32}$$

另外，α 爲所需的移動距離，也就是該次 iteration 當中要 update 權重值的幅度，由於類神經網路的誤差函數不是二次函數，因此 α 無法代二次函數的公式直接求得，可使用搜尋誤差函數值最小化的過程，來求得更新後的權重值向量。

3.2.1 Golden Section Search

搜尋函數值最小化的過程包含兩個步驟，第一個步驟為區間定位(Interval Location)，也就是沿著一個特定的方向，找尋一個包含極小值的起始區間。第二個步驟為區間縮小(Interval Reduction)，也就是將前一步驟中找到的起始區間縮小，直到滿足某一個自定的精確度為止，此處介紹的區間縮小方法為黃金分割法 [2]。

A. **Interval Location.**

如圖(3-11a)所示，如果能夠找到三個點x_a，x_b與x_c，使得$x_a < x_b < x_c$ 而且 $f(x_a) > f(x_b) < f(x_c)$，則在區間 $[x_a, x_c]$ 之間必定包含極小值。要找尋起始區間 $[x_a, x_c]$，可以使用比較函數值的方法 [2]，如圖(3-11b)所示，先計算任一個起始點x_1的函數值$f(x_1)$，接著計算與x_1相距一個固定距離ε(由使用者自定)的點x_2的函數值$f(x_2)$，繼續計算新的點x_i(與第一點之間的距離每次增加一倍)的函數值$f(x_i)$，直到連續的三次函數值呈現由大變小，再由小轉大的狀況出現為止，此三個函數值的狀況如圖(3-11b)的三個點x_2、x_3與x_4所示。區間 $[x_2, x_4]$ 即相當於圖(3-11a)裡的區間 $[x_a, x_c]$，此區間必定包含一個極小值，然而，這個區間無法再被繼續縮小，因為極小值有可能位於區間 $[x_a, x_b]$ 或是 $[x_b, x_c]$。

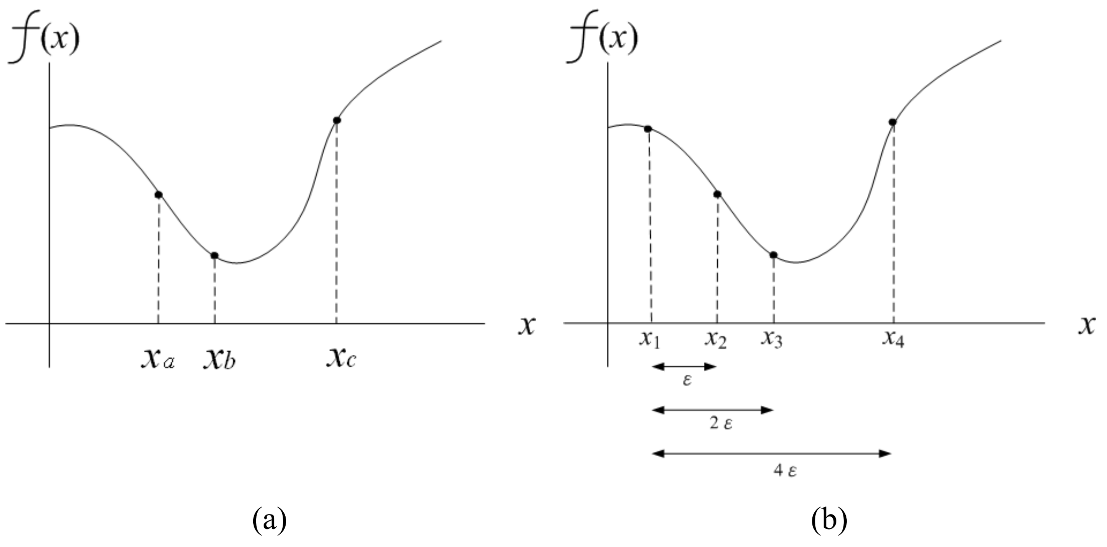


Fig. 3-11. Interval location.

以下爲區間定位的演算法形式：

Algorithm 3-2: 以區間定位方法，尋找包含函數 f 極小值的區間
Input: 函數 f 與起始搜尋點 x_1
Output: 包含極小值的區間

```
Step1: 設 $x_{start} = x_1$
       設函數計算點與點之間的距離 $\varepsilon$
       設 $x_2 = x_1 + \varepsilon$
       設 $x_3 = x_1 + 2\varepsilon$
       設 $inc = 4\varepsilon$
       計算函數值 $f(x_1)$、$f(x_2)$ 與 $f(x_3)$
Step2: WHILE ( $f(x_1) > f(x_2)$ AND $f(x_2) < f(x_3)$ 不成立)
           $f(x_1) = f(x_2)$                (儲存函數值 $f(x_2)$)
           $f(x_2) = f(x_3)$                (儲存函數值 $f(x_3)$)
           $x_1 = x_2$                      (更新搜尋點 $x_1$)
           $x_2 = x_3$                      (更新搜尋點 $x_2$)
           $x_3 = x_{start} + inc$          (更新搜尋點 $x_3$)
           計算函數值 $f(x_3)$
           $inc = 2 \times inc$
       END
Step3: 輸出區間 $[x_1, x_3]$，演算法結束
```

Example

以下以一個例子說明區間定位的步驟：輸入爲函數 $f(x) = 2x^2 - 3x + 1$ 以及起始搜尋點 x_1，輸出爲一個包含函數極小值的區間。設定起始搜尋點 $x_1 = 0$，並依 Matlab Toolbox 的初始預設值，設函數計算點與點之間的距離 $\varepsilon = 0.01$。整個疊代運算的過程如表(3-3)所示，在 iteration 6 的時候已經滿足結束搜尋的條件，也就是三個連續的函數值 $f(x_1) > f(x_2)$ 而且 $f(x_2) < f(x_3)$，輸出的結果爲區間 $[x_1, x_3] = [0.32, 1.28]$，如圖(3-12)所示。

Table 3-3. Computation process of interval location.

Iteration	x_1	x_2	x_3	$f(x_1)$	$f(x_2)$	$f(x_3)$
0	0	0.01	0.02	1	0.97	0.94
1	0.01	0.02	0.04	0.97	0.94	0.88
2	0.02	0.04	0.08	0.94	0.88	0.77
3	0.04	0.08	0.16	0.88	0.77	0.57
4	0.08	0.16	0.32	0.77	0.57	0.24
5	0.16	0.32	0.64	0.57	0.24	-0.10
6	0.32	0.64	1.28	0.24	-0.10	0.43

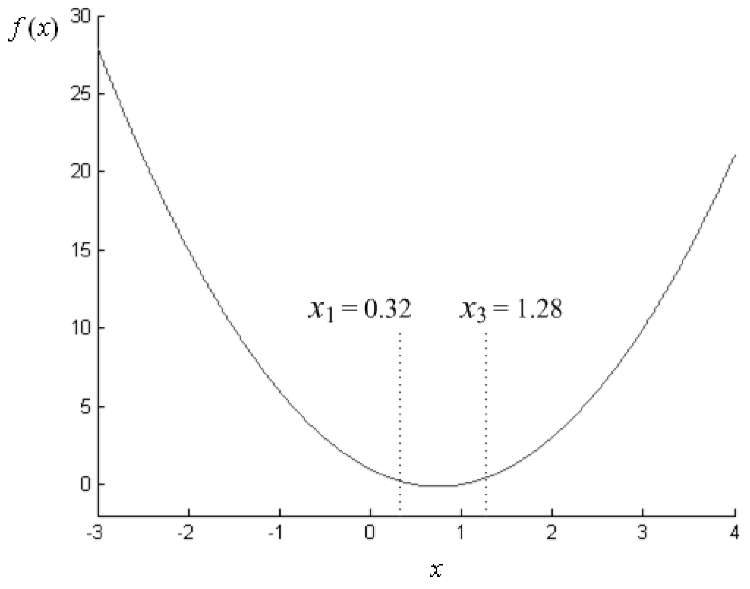


Fig. 3-12. Result of interval location.

B. Interval Reduction.

一旦包含極小值的起始區間 [x_a, x_c] 被定位出來，緊接著便利用區間縮小步驟來縮小此起始區間，直到滿足一個自定的精確度爲止。本小節介紹的區間縮小方法爲黃金分割法，一開始它需要計算兩個內部點的函數值，才能減少區間尺寸，此兩點內部點以x_1與x_2表示，滿足關係式$x_a<x_1<x_2<x_c$，並且令x_a與x_1之間的距離等於x_2與x_c之間的距離。經由比較函數值$f(x_1)$與$f(x_2)$，可以決定極小點是位於區間 [x_a, x_2] 或是 [x_1, x_c]。如圖(3-13)，

如果 $f(x_1) > f(x_2)$，則以 x_1 取代邊界點 x_a，以 x_2 取代點 x_1，新的區間變成原來的 [x_1, x_c]，另一方面，如圖(3-14)所示，如果 $f(x_1) < f(x_2)$，則以 x_2 取代邊界點 x_c，以 x_1 取代點 x_2，新的區間變成原來的 [x_a, x_2]。

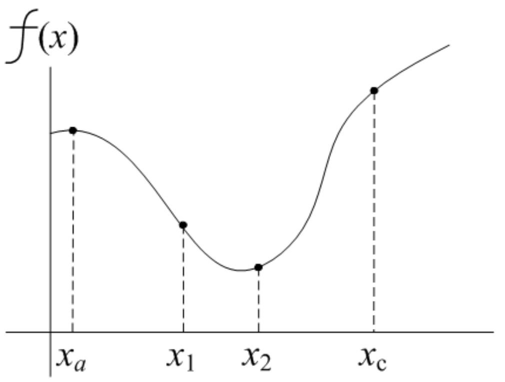


Fig. 3-13. Minimum occurs in interval [x_1, x_c].

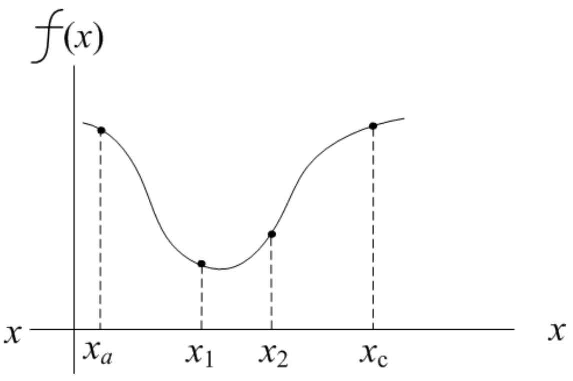


Fig. 3-14. Minimum occurs in interval [x_a, x_2].

以下的說明描述了內部點 x_1 與 x_2 的起始擺放位置，如圖(3-15)所示，假設以區間 [0, 1] 代表 [x_a, x_c]，並且以 r 表示 x_a 到 x_1 的距離與 x_a 到 x_c 的距離的比值，也就是：

$$\frac{x_1 - x_a}{x_c - x_a} = r$$

另外以 z 表示 x_1 到 x_2 的距離與 x_a 到 x_c 的距離的比值，也就是：

$$\frac{x_2 - x_1}{x_c - x_a} = z = 1 - 2r \tag{3-33}$$

如之前所述，縮小後的區間可能是 [x_a, x_2]，也可能是 [x_1, x_c]，在此處的狀況下新的區間是 [x_1, x_c]，黃金分割法的特性為在新的區間 [x_1, x_c] 中，[x_2, x_1] 的長度與 [x_1, x_c] 長度的比值，會等於原來的區間 [x_a, x_c] 中，[x_1, x_a] 的長度與 [x_a, x_c] 長度的比值，也就是：

$$\frac{z}{1-r} = \frac{r}{1} = r \tag{3-34}$$

由以上公式(3-33)與(3-34)，可以整理出：

$$\frac{1-2r}{1-r} = r$$

$$\Rightarrow\ 1 - 3r + r^2 = 0$$

$$\Rightarrow \ r = \frac{3-\sqrt{5}}{2} \approx 0.382$$

求出x_1到x_a的距離對起始區間［x_a，x_c］的比值r之後，就可以決定出兩個內部點x_1與x_2的初始位置，也就是：

$$x_1 = x_a + r(x_c - x_a)$$
$$x_2 = x_c - r(x_c - x_a)$$

由以上的說明可知，使用黃金分割法縮小區間的過程，分成兩個步驟，第一步是初始化過程，也就是在區間中放置兩個內部點，如圖(3-15)所示，[x_a，x_c]爲起始區間，x_1與x_2爲兩個初始內部點，x_1到x_a的距離等於x_2到x_c的距離；第二步則是重複縮小區間的過程，也就是在每一次 iteration 中，藉由移除一個邊界點與決定一個新的內部點的擺放位置，不斷地縮小區間直到達到預期的精確度爲止，另外，在每一次 iteration 中，黃金分割法只需計算一次函數值，即可將區間大小減小爲原本的 1-0.382 ＝ 0.618 倍。縮小區間的過程如圖(3-16)所示，於圖(3-16a)中，由於新的點x_3的函數值$f(x_3) > f(x_2)$，因此點x_c會被取代掉，新的區間變爲［x_1，x_3］，並新增一內部點x_4；於圖(3-16b)，由於$f(x_2) > f(x_4)$，因此點x_3被去掉，新的區間爲［x_1，x_2］，並新增一內部點x_5，如圖(3-16c)所示，下一步則由於$f(x_5) > f(x_4)$，x_1將會被去掉，新的區間變成［x_5，x_2］。

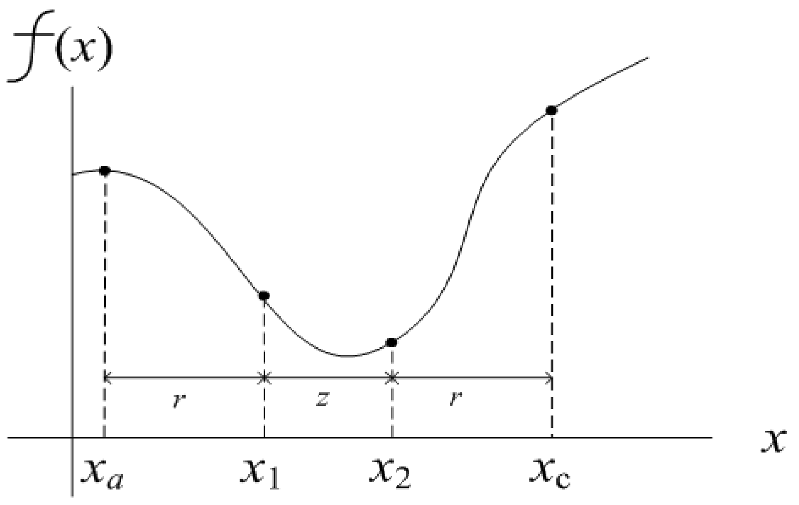


Fig. 3-15. Two initial internal points.

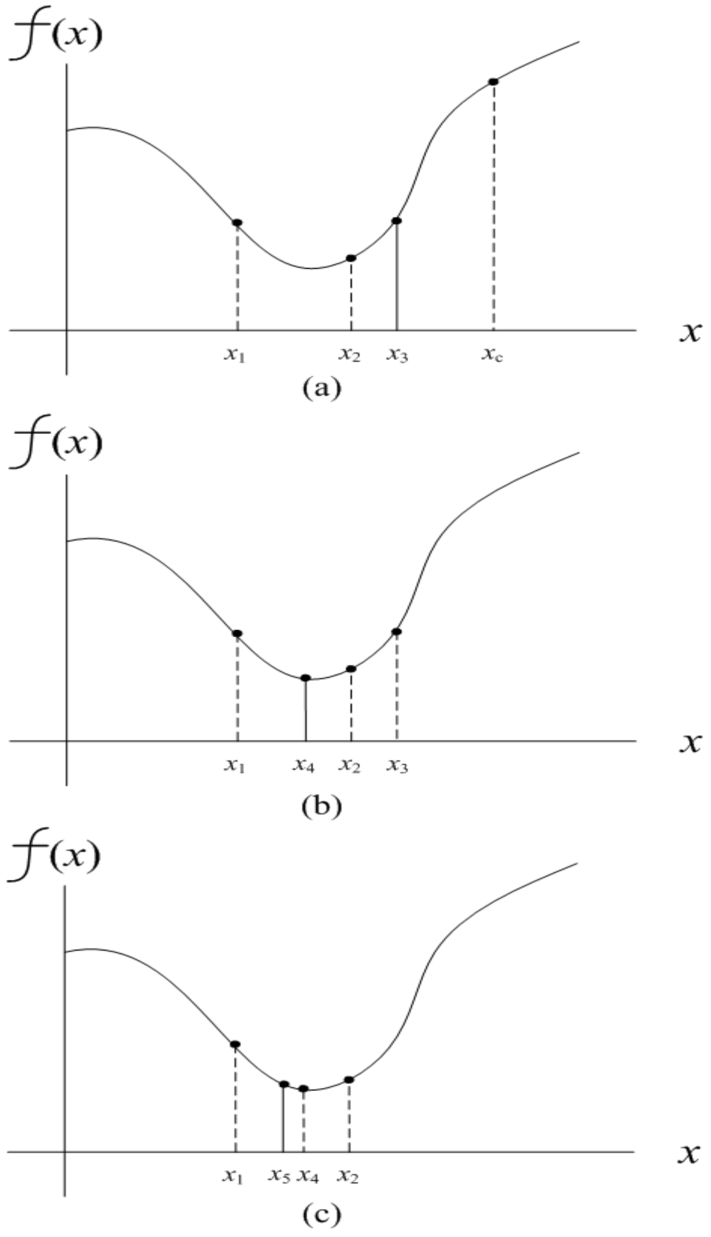
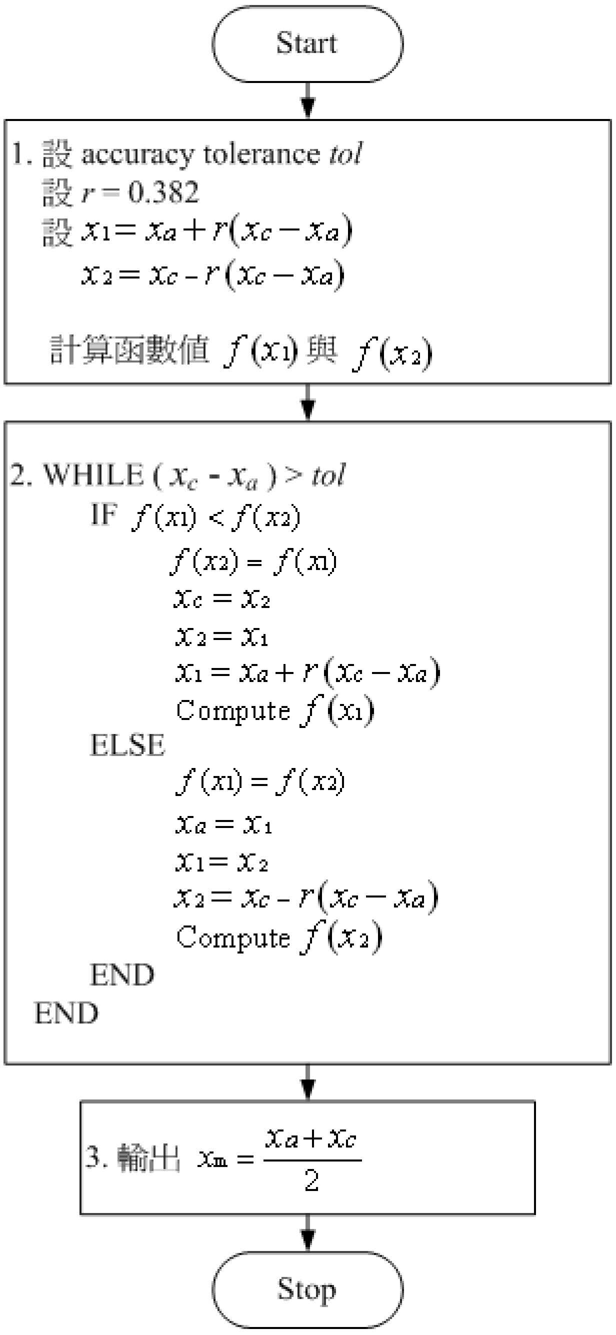


Fig. 3-16. Interval reduction.

以下為黃金分割法的演算法形式，其中區間［x_a，x_c］為經由區間定位步驟所得到的起始區間，*tol* 表示包含極小值的區間精確度，最後輸出的極小點位置以滿足此精確度的最小區間的中點表示 [14]，圖(3-17)為流程圖：

Algorithm 3-3: 以黃金分割法尋找函數 f 的極小值位置
Input: 起始區間［x_a，x_c］
函數 f
Output: 函數 f 的極小值位置x_m

```
Step 1:  設區間精確度 $tol$.
            設 $r = 0.382$
            設 $x_1 = x_a + r(x_c - x_a)$
               $x_2 = x_c - r(x_c - x_a)$
            計算函數值 $f(x_1)$ 與 $f(x_2)$
Step 2: WHILE ( $x_c - x_a$ ) > $tol$
            IF  $f(x_1) < f(x_2)$
                $f(x_2) = f(x_1)$
                $x_c = x_2$
                $x_2 = x_1$
                $x_1 = x_a + r(x_c - x_a)$
                Compute $f(x_1)$
            ELSE
                $f(x_1) = f(x_2)$
                $x_a = x_1$
                $x_1 = x_2$
                $x_2 = x_c - r(x_c - x_a)$
                Compute $f(x_2)$
            END
        END
Step 3:輸出$x_m = \frac{x_a + x_c}{2}$，演算法停止
```



```
1. 設 accuracy tolerance $tol$
   設 $r = 0.382$
   設 $x_1 = x_a + r(x_c - x_a)$
      $x_2 = x_c - r(x_c - x_a)$

   計算函數值 $f(x_1)$ 與 $f(x_2)$

2. WHILE ( $x_c$ - $x_a$ ) > $tol$
       IF $f(x_1) < f(x_2)$
           $f(x_2) = f(x_1)$
           $x_c = x_2$
           $x_2 = x_1$
           $x_1 = x_a + r(x_c - x_a)$
           Compute $f(x_1)$
       ELSE
           $f(x_1) = f(x_2)$
           $x_a = x_1$
           $x_1 = x_2$
           $x_2 = x_c - r(x_c - x_a)$
           Compute $f(x_2)$
       END
   END

3. 輸出 $x_m = \frac{x_a + x_c}{2}$
```

Fig. 3-17. Flowchart of Golden Section Search.

Example

以下延續上一小節區間定位步驟的範例，繼續以黃金分割法來找出函數極小值的位置，輸入為 $f(x) = 2x^2 - 3x + 1$，起始區間為 [0.32, 1.28]，並依Matlab Toolbox的初始預設值將區間精確度*tol*設為 0.0005，整個疊代運算的過程如表(3-4)所示，可以發現在iteration 16 的時候，區間 [x_a, x_c] = [0.7498, 0.7503]，已

經到達精確度要求，因此演算法結束，並輸出函數最小值的位置 $x_m = \frac{0.7498 + 0.7503}{2} \approx 0.75$ 。

Table 3-4. Computation process of *Golden Section Search*.

Iteration	x_a	x_c	x_1	x_2	$f(x_1)$	$f(x_2)$
0	0.32	1.28	0.6867	0.9133	-0.1170	-0.0717
1	0.32	0.9133	0.5466	0.6867	-0.0423	-0.1170
2	0.5466	0.9133	0.6867	0.7732	-0.1170	-0.1239
3	0.6867	0.9133	0.7732	0.8267	-0.1239	-0.1132
4	0.6867	0.8267	0.7402	0.7732	-0.1248	-0.1239
5	0.6867	0.7732	0.7198	0.7402	-0.1232	-0.1248
6	0.7198	0.7732	0.7402	0.7528	-0.1248	-0.1250
7	0.7402	0.7732	0.7528	0.7606	-0.1250	-0.1248
8	0.7402	0.7606	0.7480	0.7528	-0.1250	-0.1250
9	0.7402	0.7528	0.7450	0.7480	-0.1250	-0.1250
10	0.7450	0.7528	0.7480	0.7498	-0.1250	-0.1250
11	0.7480	0.7528	0.7498	0.7510	-0.1250	-0.1250
12	0.7480	0.7510	0.7491	0.7498	-0.1250	-0.1250
13	0.7491	0.7510	0.7498	0.7503	-0.1250	-0.1250
14	0.7491	0.7503	0.7496	0.7498	-0.1250	-0.1250
15	0.7496	0.7503	0.7498	0.7500	-0.1250	-0.1250
16	0.7498	0.7503	0.7500	0.7501	-0.1250	-0.1250

3.2.2 Training Process Using Conjugate Gradient Method

由以上的討論可以整理出，以共軛梯度倒傳遞演算法，應用於訓練類神經網路的過程，就是在每一次 iteration 中，首先要計算搜尋的方向，然後再沿著此搜尋方向，尋找要更新的權重值向量，其中，搜尋的方向是經由公式(3-29)或(3-31)計算得來，而更新的權重值向量則可由黃金分割法計算而來。

當應用黃金分割法於類神經網路時，同樣依照區間定位與區間縮小兩步驟，來尋找更新的權重值向量。區間定位步驟是從目前的權重值向量 **w** 開始，計算該點的網路誤差函數值 E(**w**)，接著是計算第二點的網路誤差函數值，它與

第一點的位置距離爲 ε，且沿著目前的搜尋方向 **d**，也就是計算 E(**w**+ε**d**)的值，繼續計算下一步的誤差函數值 E(**w**+2ε**d**)，這是因爲與第一步之間的距離每次皆增加一倍，因此，計算下一步的誤差函數值爲 E(**w**+4ε**d**)，如此繼續下去，直到連續的三次誤差函數值呈現由大變小，再由小轉大的狀況出現爲止，也就是包含極小值的區間被定位出來，此區間在此以 $[\mathbf{w}+\alpha_a\mathbf{d},\ \mathbf{w}+\alpha_c\mathbf{d}]$ 表示。

接著利用黃金分割法，尋找最佳的更新權重值，也就是在上一個步驟所定位出的起始區間範圍內，以目前的權重值向量**w**，沿著選定的方向**d**，找尋移動 α 距離後的權重值向量，使得網路誤差函數值E(**w**＋α **d**) 在此方向**d**上最小化。以下爲應用黃金分割法，找出最佳的更新後的權重值向量的演算法程序。

```
Algorithm 3-4: 應用黃金分割法，找出更新的類神經網路權重值向量
Input: 起始區間 [ $\mathbf{w}+\alpha_a\mathbf{d}$ , $\mathbf{w}+\alpha_c\mathbf{d}$ ]
       目前的權重值向量 w 與搜尋方向 d
       網路的誤差函數 E (公式 3-26)
Output:更新的權重值向量

Step 1: 設定精確度 tol.
        設 $r = 0.382$
        設 $\alpha_1 = \alpha_a + r(\alpha_c - \alpha_a)$
           $\alpha_2 = \alpha_c - r(\alpha_c - \alpha_a)$
        計算誤差函數值E($\mathbf{w}+\alpha_1\mathbf{d}$)與E($\mathbf{w}+\alpha_2\mathbf{d}$)
Step 2: WHILE ( $\alpha_c - \alpha_a$ ) > tol
            IF E($\mathbf{w}+\alpha_1\mathbf{d}$) < E($\mathbf{w}+\alpha_2\mathbf{d}$)
                E($\mathbf{w}+\alpha_2\mathbf{d}$) = E($\mathbf{w}+\alpha_1\mathbf{d}$)
                $\alpha_c = \alpha_2$
                $\alpha_2 = \alpha_1$
                $\alpha_1 = \alpha_a + r(\alpha_c - \alpha_a)$
                Compute E($\mathbf{w}+\alpha_1\mathbf{d}$)
            ELSE
                E($\mathbf{w}+\alpha_1\mathbf{d}$) = E($\mathbf{w}+\alpha_2\mathbf{d}$)
                $\alpha_a = \alpha_1$
                $\alpha_1 = \alpha_2$
                $\alpha_2 = \alpha_c - r(\alpha_c - \alpha_a)$
                Compute E($\mathbf{w}+\alpha_2\mathbf{d}$)
            END
```

```
    END
Step 3: 取 $\alpha = \dfrac{\alpha_a+\alpha_c}{2}$，輸出更新後的權重值向量 $\mathbf{w}+\alpha\mathbf{d}$，並停止演算法
```

以下爲以共軛梯度倒傳遞演算法訓練類神經網路的演算法流程，圖(3-18)爲流程圖。

Algorithm 3-5: 使用共軛梯度倒傳遞演算法訓練類神經網路

Input: 訓練樣本以及期望輸出值

誤差函數 E 用以計算網路 error

Output: 訓練完成的網路權重值

```
Step 1: 設 index $t = 0$, 以及執行的疊代次數 T = 0
        設定 stopping error 以及最大疊代數
        給定一個初始化的權重值向量 $\mathbf{w}(0)$
Step 2: 輸入訓練樣本.
        計算梯度向量 $\mathbf{g}(t)$：
            對輸出層單元，使用以下公式：
                $g_{kj} = -\frac{1}{N}\sum_{n=1}^{N}[\,(d_{nk}-y_{nk})f'(net_{nk})y_{nj}\,]$
            對隱藏層單元，使用以下公式：
                $g_{ji} = -\frac{1}{N}\sum_{n=1}^{N}\left\{[\sum_{k=1}^{K}(d_{nk}-y_{nk})f'(net_{nk})w_{kj}]f'(net_{nj})y_{ni}\right\}$
Step 3: 計算搜尋方向：
            IF $t = 0$
                $\mathbf{d}(0) = -\mathbf{g}(0)$
            ELSE
                $\beta(t) = \frac{\mathbf{g}(t)^{T}\mathbf{g}(t)}{\mathbf{g}(t\text{-}1)^{T}\mathbf{g}(t\text{-}1)}$
                $\mathbf{d}(t) = -\mathbf{g}(t) + \beta(t)\mathbf{d}(t\text{-}1)$
            END
Step 4: 根據目前的權重值向量$\mathbf{w}(t)$與搜尋方向$\mathbf{d}(t)$，使用黃金分割法找出更新
```

的權重值向量；即先以區間定位步驟找出起始區間 $[\alpha_a, \alpha_c]$，接著再以目前的權重值向量$\mathbf{w}(t)$，沿著目前選定的方向$\mathbf{d}(t)$，在此起始區間範圍內，找尋移動$\alpha(t)$距離後的權重值向量，使得網路誤差函數值$E(\mathbf{w}(t) + \alpha(t)\mathbf{d}(t))$ 最小化。

Step 5: 調整成新的權重值向量：

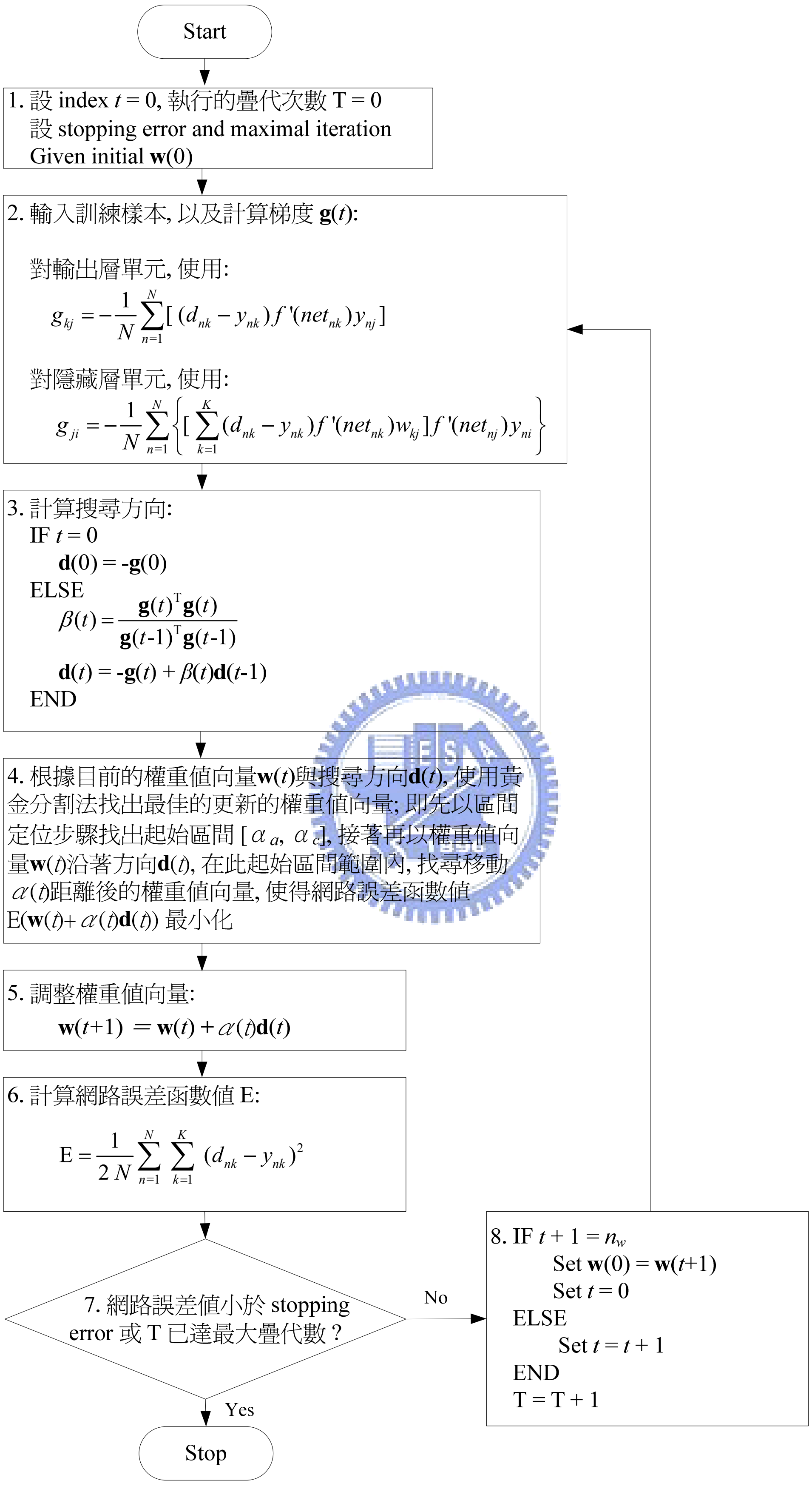
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha(t)\mathbf{d}(t)$$

Step 6: 使用以下公式計算 error，以便測試網路：

$$E = \frac{1}{2N}\sum_{n=1}^{N}\sum_{k=1}^{K}(d_{nk} - y_{nk})^2$$

Step 7: 如果 error 小於 stopping error 或者 T 已到達最大量代數，則停止；否則進行下一步

```
Step 8: IF $t+1 = n_w$ ($n_w$是整個網路權重值係數的總數)
            設 $\mathbf{w}(0) = \mathbf{w}(t+1)$
            設 $t = 0$
        ELSE
            設 $t = t + 1$
        END
        $T = T + 1$
        Repeat by going to step 2
```



Start

1. 設 index $t = 0$, 執行的疊代次數 T = 0
   設 stopping error and maximal iteration
   Given initial $\mathbf{w}(0)$

2. 輸入訓練樣本, 以及計算梯度 $\mathbf{g}(t)$:

   對輸出層單元, 使用:

$$g_{kj} = -\frac{1}{N}\sum_{n=1}^{N}[\,(d_{nk} - y_{nk})f'(net_{nk})y_{nj}\,]$$

   對隱藏層單元, 使用:

$$g_{ji} = -\frac{1}{N}\sum_{n=1}^{N}\left\{[\,\sum_{k=1}^{K}(d_{nk} - y_{nk})f'(net_{nk})w_{kj}\,]f'(net_{nj})y_{ni}\right\}$$

3. 計算搜尋方向:

```
IF $t = 0$
    $\mathbf{d}(0) = -\mathbf{g}(0)$
ELSE
    $\beta(t) = \dfrac{\mathbf{g}(t)^{\mathrm{T}}\mathbf{g}(t)}{\mathbf{g}(t\text{-}1)^{\mathrm{T}}\mathbf{g}(t\text{-}1)}$
    $\mathbf{d}(t) = -\mathbf{g}(t) + \beta(t)\mathbf{d}(t\text{-}1)$
END
```

4. 根據目前的權重值向量$\mathbf{w}(t)$與搜尋方向$\mathbf{d}(t)$, 使用黃金分割法找出最佳的更新的權重值向量; 即先以區間定位步驟找出起始區間 $[\alpha_a, \alpha_c]$, 接著再以權重值向量$\mathbf{w}(t)$沿著方向$\mathbf{d}(t)$, 在此起始區間範圍內, 找尋移動 $\alpha(t)$距離後的權重值向量, 使得網路誤差函數值 $\mathrm{E}(\mathbf{w}(t)+\alpha(t)\mathbf{d}(t))$ 最小化

5. 調整權重值向量:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha(t)\mathbf{d}(t)$$

6. 計算網路誤差函數值 E:

$$\mathrm{E} = \frac{1}{2N}\sum_{n=1}^{N}\sum_{k=1}^{K}(d_{nk} - y_{nk})^2$$

7. 網路誤差值小於 stopping error 或 T 已達最大疊代數 ?

No

```
8. IF $t + 1 = n_w$
        Set $\mathbf{w}(0) = \mathbf{w}(t+1)$
        Set $t = 0$
   ELSE
        Set $t = t + 1$
   END
   T = T + 1
```

Yes

Stop

Fig. 3-18. Flowchart of CGBP algorithm to train MLP network.

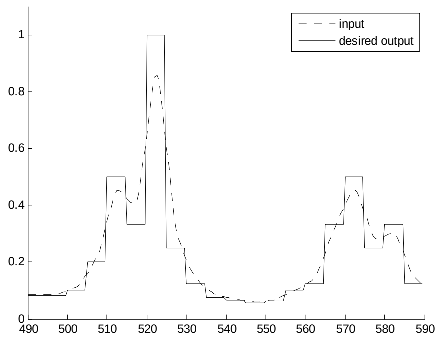
4. Experiments on Well Logging Inversion

We use the well logging datasets in general MLP network and higher-order feature neural network that is trained by gradient descent method and conjugate gradient method respectively. From section 4.2 to section 4.5, the used learning rule is gradient descent method. In section 4.6, the used learning rule is conjugate gradient method. The input to neural network is the conductivity, which is the inverse of combined synthetic apparent resistivity, and the inverted true formation conductivity is the network output. The total number of well logging dataset is 31 and each of them contains 200 input/output pairs. The depth is from 490 feet to 589.5 feet, and data is recorded every 0.5 feet. Three of the datasets are shown in Figure 4-1.

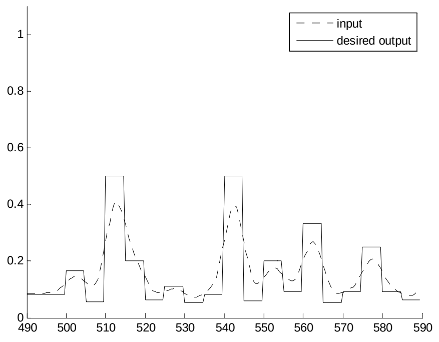
We do experiments with Matlab ® v7.1, using the personal computer with single Intel ® 1.8 GHz processor and 512 Mb of memory. For each experiment, we use 30 datasets as the training set to train the network, and the remaining 1 dataset is used to evaluate the performance of trained network in each trial. After 31 trials, average these 31 testing results as the experimental performance. The performance in each trial is evaluated by comparing the real outputs that are inverted by network and the desired outputs of testing dataset, so we use the mean absolute error (MAE) to evaluate the performance of a trained network.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} | d_i - y_i |$$

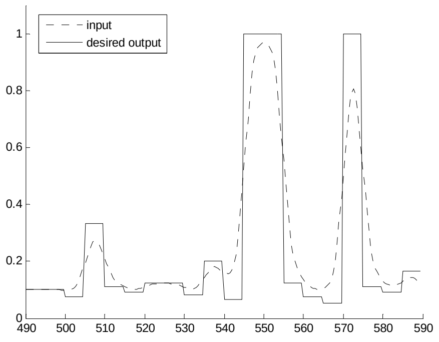
where N is the number of features, $N = 200$, d_i and y_i is the desired output and the real output of the i-th node.



(a) Dataset number 6.



(b) Dataset number 16.



(c) Dataset number 31.

Fig. 4-1. Three samples of well logging datasets.

4.1 Data Pre-processing

A data normalization process is carried out in pre-processing phase. In our experiments, the data is normalized to range [0.1, 0.9]. After testing, the testing value is then reversed back to the original data range. Supposed the largest value in dataset is denoted as *M*, the smallest value is denoted as *N*, then arbitrary value *x* that in range [N, M] can be transformed to *x*' that in range [*a, b*] by

$$x' = [(x - N) (b - a) / (M - N)] + a$$

and is reversed back to the original range [N,M] by

$$y' = [(y - a) (M - N) / (b - a)] + N$$

4.2 Determination of the Stopping Error

In this section, we compare the network performance with different stopping errors. The architecture of MLP network we used is 200-400-200 (not include bias). The learning rate is set to 0.1, the momentum parameter is set to 0.9, and the maximal iteration is set to 10,000. To decrease the impact of the initial weight, we repeat 15 trials for each testing case. In each trial, 30 datasets are used for training and 1 dataset is used for testing.

First, the used stopping errors are set to 0.000005, 0.00001 and 0.000015 respectively. The testing result shows that the mean absolute error of network when stopping error is set to 0.000005 is the smallest one, however, the training time is apparently longer. On the other hand, the difference of training time between the network using stopping error of 0.00001 and 0.000015 is smaller but the difference of mean absolute is about 2.48%, so we consider the stopping error of 0.00001 is appropriate. The testing result is shown in Table 4-1(a). Second, we compare the network performance using the stopping error around 0.00001, which range from 0.000008 to 0.000012. The testing result is shown in Table 4-1(b). There is no significant difference about absolute error between them. By considering the mean absolute error and the training time, we choose the stopping error of 0.00001.

Table 4-1. The network performance using different stopping errors.

(a) The used stopping error of 0.000005, 0.00001, and 0.000015.

Stopping error	Avg. of MAE	Avg. iterations	Avg. of training time (Sec.)
0.000005	0.096507	2,565	8,134
0.000010	0.116438	1,658	4,467
0.000015	0.141254	1,159	3,710

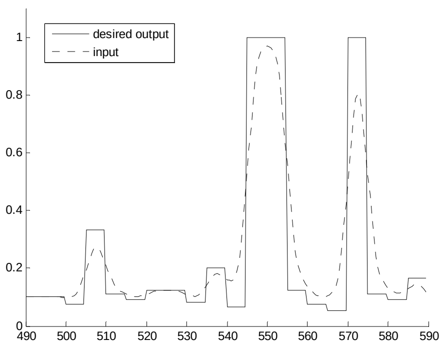
(b) The used stopping error from 0.000008 to 0.000012.

Stopping error	Avg. of MAE	Avg. iterations	Avg. of training time (Sec.)
0.000008	0.118764	2,243	6,487
0.000009	0.120594	2,016	5,461
0.000010	0.116438	1,658	4,467
0.000011	0.128648	1,562	4,665
0.000012	0.127716	1,348	4,250

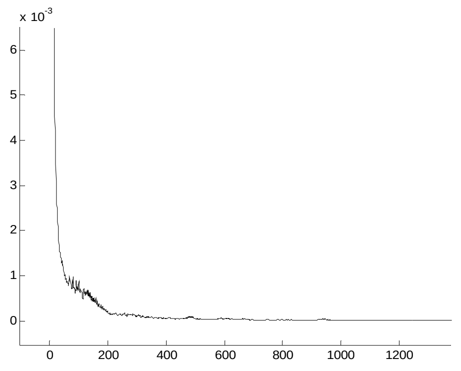
4.3 Determination of the Number of Input Nodes

A. MLP Network with 200 Input and Output Nodes

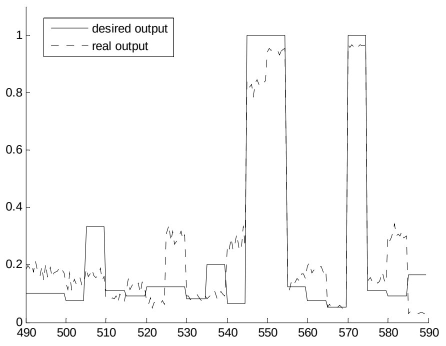
In the network architecture of 200-400-200, the number of connection weights of such a network is 160,600 (201x400+401x200, including bias), and divergence is happened in our experiment sometimes. One of the experiments that successfully achieve the convergence is shown in Figure 4-2. In that experiment we use dataset number 1 to number 30 as the training datasets, and number 31 as the testing dataset. Figure 4-2(a) shows the figure of well logging dataset umber 31, including the inputs and the desired outputs. Figure 4-2(b) shows the MSE curve in training, and Figure 4-2(c) shows the testing result, which the solid line is the desired outputs and the dotted line is the real outputs. The number of training iterations in this experiment is 3,379, the CPU time is 5,853 seconds, and the mean absolute error is 0.0895. One can see that the fitting result is not well, so we will try to increase the number of training pattern to improve the network performance in next section.



(a) Original well logging dataset 31.



(b) MSE curve in training.



(c) Testing result.

Fig. 4-2. Testing result of MLP network with 200 input nodes.

B. Network with Smaller Number of Input and Output Nodes

In order to increase the number of training pattern and the convergence rate, we use the smaller network by grouping the input features of each dataset. The number of input nodes of network is set to 10, 20, 40, 100 and 200. Figure 4-3 is an example that the number of input nodes is 10. We use the features from the 1st to 10th as the first input pattern, and then use the 11th to 20th features as the second input pattern. In this way all of the features in each dataset can be processed. In this case, the number of training pattern becomes 20 for each dataset and then the total number of training patterns becomes 600 (20x30 datasets).

To evaluate the MLP network performance of different number of input nodes, 31 trials for each network are done. The learning rate is set to 0.1, the momentum parameter is 0.9, the maximal number of iterations is 10,000, and the stopping error is set to 0.00001. The number of hidden nodes of each MLP network is 20, 40, 80, 200, and 400 respectively. Table 4-2 and Figure 4-4 show the experimental results of each MLP network. The MLP network with 10 input nodes has the smallest average of mean absolute error.

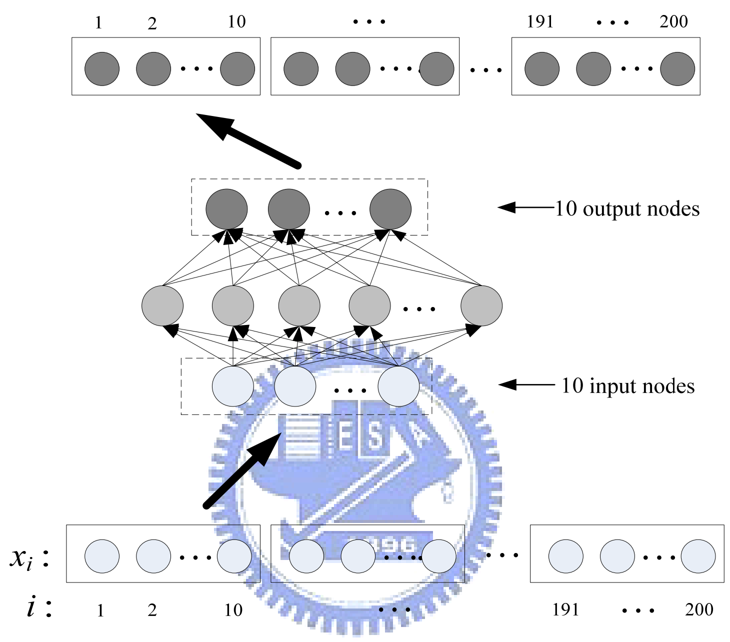


Fig. 4-3. MLP network with smaller number of input nodes.

Table 4-2. Experimental results of each kind of MLP network.

Network size	Num. of training patterns	Avg. of MAE	Avg. iterations	Avg. training time (sec.)
10-20-10	600	0.002612	1,986	2,887
20-40-20	300	0.003592	1,553	2,355
40-80-40	150	0.007378	3,230	3,465
100-200-100	60	0.061248	3,605	4,618
200-400-200	30	0.123413	1,723	4,745

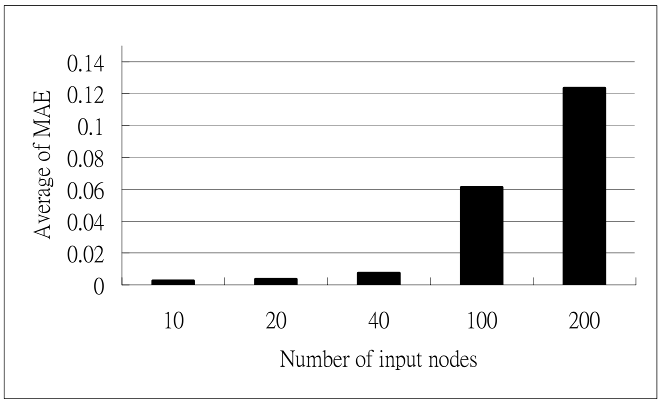


Fig. 4-4. Average performance with different number of input nodes.

4.4 Determination of the MLP Network Parameters

A. Learning Rate Determination

One of the important parameters to MLP network performance is the learning rate. The learning rate determines the speed of training. In general, a larger learning rate generates more weight change while adjusting the connection weight, but may result in the divergence because of oscillation. A smaller learning rate, on the other hand, generates a smaller weight adjusting and results in slow training speed. We use the network with 10 input nodes, 20 hidden nodes, and 10 output nodes to test with the different learning rate. The stopping error is set to 10^{-5}, the maximal number of iteration is 10,000, and the momentum parameter is set to $1-\eta$, where η denotes the learning rate. 31 trials are done for average performance of each learning rate. The testing result is shown in Table 4-3 and plot in Figure 4-5. The MLP network with learning rate 0.6 generates the smallest average of mean absolute error in our experiments.

Table 4-3. MLP network performance with different learning rate.

Network size	Num. of training patterns	Learning rate (η)	Avg. of MAE	Smallest MAE
10-20-10	600	0.1	0.002649	0.001685
	600	0.2	0.002804	0.001898
	600	0.3	0.002774	0.001910
	600	0.4	0.002873	0.001671
	600	0.5	0.002806	0.001882
	600	0.6	0.002508	0.001584
	600	0.7	0.003255	0.002096
	600	0.8	0.003184	0.002418
	600	0.9	0.004117	0.002302

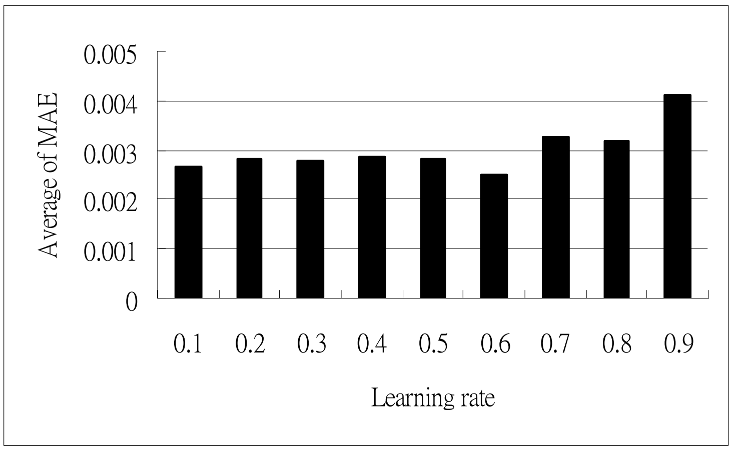


Fig. 4-5. Mean absolute error in MLP networks with different learning rate.

B. Number of Hidden Nodes Determination

For neural network architecture design, the MLP network size is usually dependent on different problem. In our experiment, the number of nodes in the input and the output layer is determined in advance, however, the choice of the number of nodes in hidden layer is not determined. A number of techniques have been developed to help selecting the appropriate number of nodes in hidden layer. One method is called pruning method [15]. The pruning method starts with a trained network with large number of hidden nodes and then tries to remove some of the redundant hidden nodes. This involves computing a saliency measure of each hidden node. A saliency measure is the importance of a hidden node to the network. A low saliency means that the hidden node has low influence to the network performance, and then can be removed. In this study, we adopt the pruning method called Unit Selection Algorithm (USA) that was proposed by Messer et al. [16] to search for the appropriate number of nodes in hidden layer.

The algorithm to test saliency needs two datasets: a training dataset D_{tr} and a verification dataset D_{ver}. We must specify a mean absolute error level E_{el} that we are ready to accept on the verification dataset.

The unit selection algorithm then proceeds as follows. The flowchart is shown in Figure 4-6.

Algorithm 4-1: Unit selection algorithm to prune MLP network.

Input: Training datasets and verification datasets.

Output: Pruned MLP network.

Step 1: Set error level E_{el}.

Step 2: Construct a large MLP network with h hidden nodes and train until the mean absolute error on training dataset is less than an error threshold.

Step 3: Calculate the saliency of each hidden node S_i, $i = 1, \ldots, h$.

$$S_i = \sum_{j=1}^{n^{(l+1)}} (w_{ji})^2 \;\; / \;\; n^{(l+1)}$$

where S_i is the saliency of node i in hidden layer l, w_{ji} is the weight connecting node i in hidden layer l to the node j in the next layer (l+1), $n^{(l+1)}$ is the number of nodes in layer (l+1). Figure 4-7 shows layer l and layer l+1 of the MLP network.

Step 4: Find the node that has the smallest saliency and remove it, and set the current number of hidden nodes $h = h - 1$.

Step 5: Retrain the network for N_r iterations and compute the mean absolute error E_{ver} on verification dataset.

Step 6: If $E_{ver} > E_{el}$, algorithm stops, else, go to step 3.

Step 7: set $h = h + 1$.

In step 6, we compare E_{ver} and error level E_{el}. If E_{ver} is larger than E_{el}, the algorithm stops, otherwise, repeat by going to step 3. In step 7, the node that just removed in step 4 must be restored, so we set $h = h + 1$. The reason is that E_{ver} will increase as more nodes are removed, so the optimal number of hidden nodes is chosen at the point just before E_{ver} exceeds the error level that we are ready to accept.

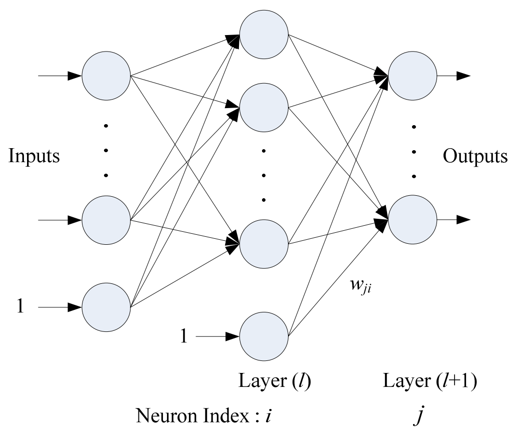


Fig. 4-6. The MLP network.

In our experiments, we construct a MLP network with 10 input nodes and 10 output nodes. Since the algorithm starts from a large network, we use 20 hidden nodes. The used learning rule is gradient descent method, the learning rate is 0.6, and the momentum parameter is 0.4. The mean absolute error level E_{el} that we are ready to accept is set to 0.005 for all experiments. Initially we train the MLP network until the mean absolute error on training dataset is less than 0.003. Then after each hidden node removal, the network is retrained for 10 iterations. Training datasets are 16 out of the total 31 datasets, and the remaining 15 datasets are used for verification. We repeat 20 experiments to find the average number of hidden nodes. Table 4-4 shows the results of 20 experiments, and the average number of hidden nodes is 11.8.

Then we compare the network performance between the original MLP network (20 hidden nodes) and the MLP network pruned by the algorithm (12 hidden nodes). 31 trials are done for each network. The number of input nodes is 10, the stopping error is set to 10^{-5}, the learning rate is 0.6, and the momentum parameter is 0.4. The testing results are shown in Table 4-5. The size of MLP network that pruned by USA algorithm is smaller but still keeps the same performance level. However, the used training time is different. A smaller-size network not only saves the required memory but also saves the training time. The suggested number of hidden nodes is 1.2 times of the number of input nodes for the experiments.

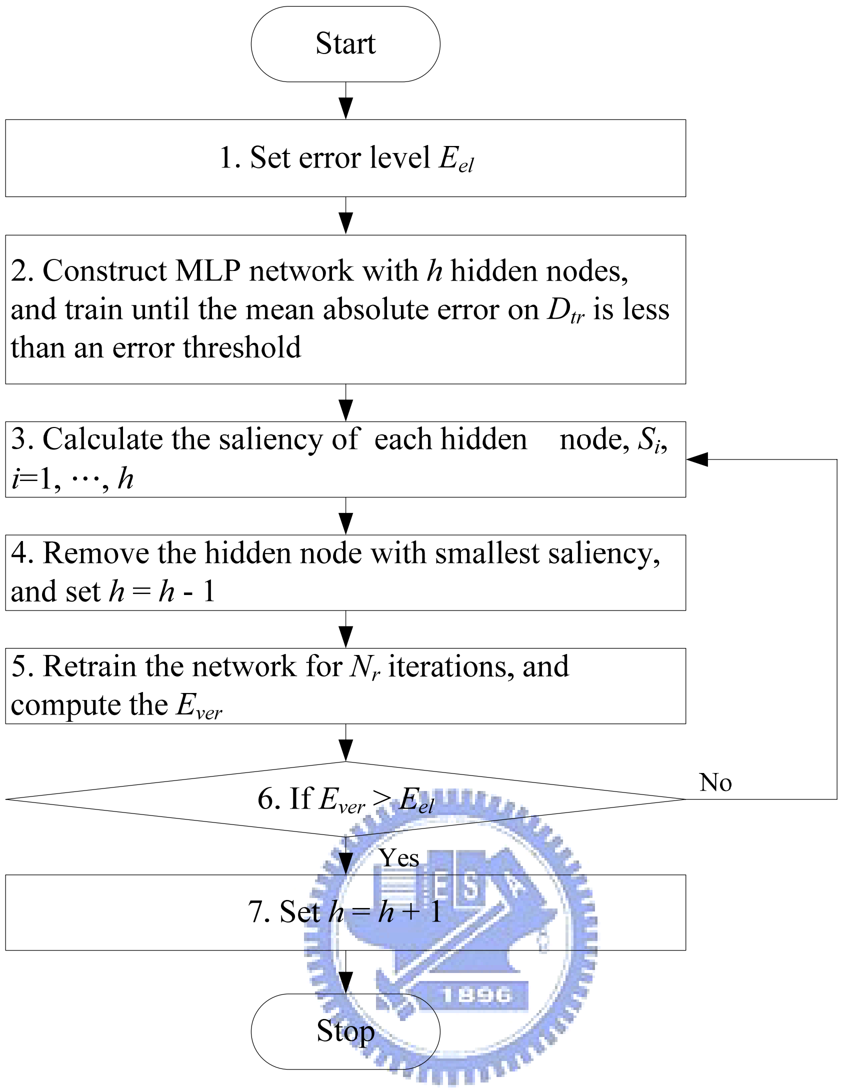

Start
1. Set error level E_{el}
2. Construct MLP network with h hidden nodes, and train until the mean absolute error on D_{tr} is less than an error threshold
3. Calculate the saliency of each hidden node, S_i, i=1, ⋯, h
4. Remove the hidden node with smallest saliency, and set $h = h - 1$
5. Retrain the network for N_r iterations, and compute the E_{ver}
6. If $E_{ver} > E_{el}$
No
Yes
7. Set $h = h + 1$
Stop

Fig. 4-7. Flowchart of USA algorithm.

Table 4-4. Number of hidden nodes by USA algorithm.

Hidden nodes by USA algorithm in 20 experiments	12, 14, 11, 9, 10, 16, 14, 13, 10, 11, 13, 16, 13, 12, 8, 12, 9, 10, 12, 11.
Average	11.8

Table 4-5. Comparison of MLP network performance between full network and the network that pruned by USA algorithm.

	Network size	Number of training patterns	Avg. of MAE	Smallest MAE	Avg. training time (Sec.)
Full network	10-20-10	600	0.002972	0.001904	2,454
USA	10-12-10	600	0.003008	0.002065	2,105

C. Number of Hidden Layers Determination

We use the second hidden layer and test the MLP networks with different number of nodes in the second hidden layer. 31 trials are done for each MLP network, and the number of training datasets is 30 for each trial The network parameters are the same as used in the previous experiments that the learning rate is 0.6, the momentum parameter is 0.4, and the stopping error is 10^{-5}. Table 4-6 shows the experimental results of network with one and two hidden layers. For both the average of mean absolute error and the smallest error cases, no significant difference can be observed. Adding one more hidden layer does not provide apparent improvement, so we use one hidden layer in our experiments.

Table 4-6. Comparison of MLP network performance with one and two hidden layers.

Hidden layer(s)	Network size	Number of training patterns	Avg. of MAE	Smallest MAE	Avg. training time (sec.)
One hidden layer	10-12-10	600	0.003008	0.002065	2,105
Two hidden layers	10-12-3-10	600	0.003169	0.002258	2,602
	10-12-4-10	600	0.003304	0.002323	1,957
	10-12-5-10	600	0.003312	0.002257	2,751
	10-12-6-10	600	0.003128	0.002487	2,446
	10-12-7-10	600	0.003058	0.002037	1,954

4.5 Higher-Order Feature Multi-Layer Neural Net (HOML)

In order to enhance the non-linear mapping of network, input features can be expanded to a larger space through the transformation of non-linear functions. In this study, we expand the input features using the higher-order function which from second-order function to 5th-order function. Using these functions we design 5 higher-order feature neural nets: HOML-1, HOML-2, HOML-3, HOML-4, and HOML-5, and are listed in Table 4-7. HOML-1 is actually the general MLP network with no higher-order features. All of these 5 network types are trained by gradient descent method, and the number of hidden nodes is 1.2 times of the number of input nodes. Figure 4-8 is an example that shows the HOML-2 that expands input features using second-order function.

Table 4-7. Five different HOML types.

Network type	Features (include bias)	Network size
HOML-1	$1+x$	10-12-10
HOML-2	$1+x+x^2$	20-24-10
HOML-3	$1+x+x^2+x^3$	30-36-10
HOML-4	$1+x+x^2+x^3+x^4$	40-48-10
HOML-5	$1+x+x^2+x^3+x^4+x^5$	50-60-10

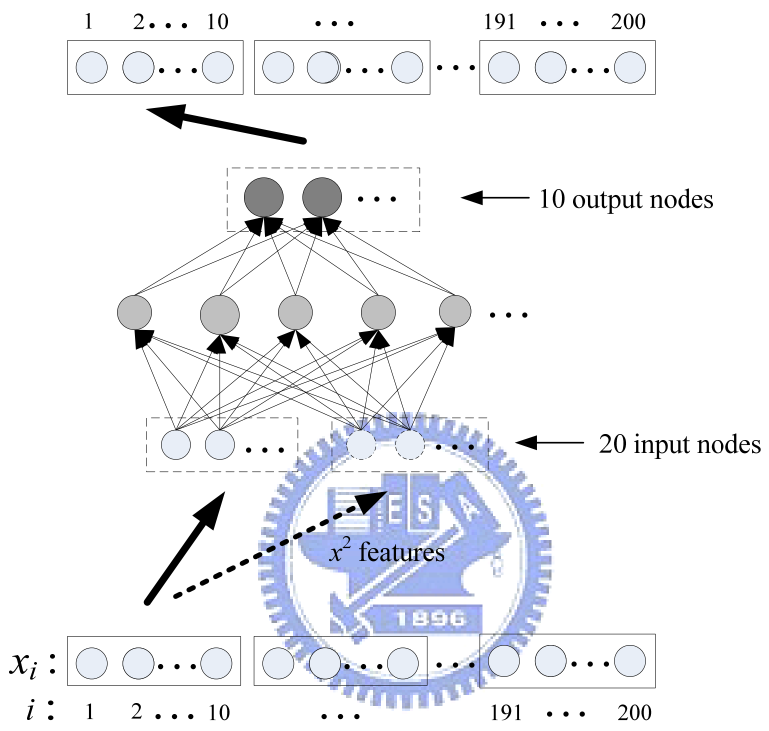


Fig. 4-8. HOML-2 with original and expanded features.

A. Comparison of Performance of Network that without Hidden Layer and that with One Hidden Layer

In this section, we use five higher-order feature neural nets with single layer: HOSL-1, HOSL-2, HOSL-3, HOSL-4, and HOSL-5 that listed in Table 4-8. Figure 4-9 is an example that shows the HOSL-2 that expands input features using second-order function. We test the inverse ability of HOSL networks for well logging data and compare the performance with the HOML networks. Both HOSL and HOML networks use gradient descent method as learning rule. The learning rate is 0.6, the momentum parameter is 0.4, and the stopping error is 0.00001. The maximum iterations are set to 10,000. Repeat 10 trials for each experiment and the number of training datasets is 30 for each trial. The dataset number to test the trained network for these 10 trials is 1, 5, 10, 12, 15, 17, 20, 23, 25, and 30 respectively. Each experimental result is the average of 10 trials and is shown in Table 4-9. The testing results show that higher-order feature neural nets without hidden layers can be used for well logging data inversion. However, from the results we found:

First, the average training time of each HOSL network is long since they failed to achieve the training goal and stop training at maximum iterations. However, HOSL-3, 4, and 5 have significant improvement in mean absolute error. The reason is that the networks become more non-linear with the usage of higher-order features.

Second, for each network type, the average training time of HOML is shorter than that of HOSL. This is because the HOML networks are more non-linear than HOSL networks with the usage of hidden layer and converged before reaching the maximal iterations. Figure 4-10 shows the comparison of MSE curve between HOML-5 and HOSL-5, the used testing dataset is number 30. HOSL-5 is failed to achieve the training goal and reaches the maximal iterations. The training time of HOML-5 is shorter, and the training goal is achieved at 1,189 iterations. From the above discussion we found that the network with one hidden layer performs better than that without hidden layer, so we use the one-hidden layer network to test the well logging data inversion.

Table 4-8. Five different HOSL types.

Network type	Features (include bias)	Network size
HOSL-1	$1 + x$	10-10
HOSL-2	$1 + x + x^2$	20-10
HOSL-3	$1 + x + x^2 + x^3$	30-10
HOSL-4	$1 + x + x^2 + x^3 + x^4$	40-10
HOSL-5	$1 + x + x^2 + x^3 + x^4 + x^5$	50-10

Table 4-9. Testing results of HOSL and HOML.

	Network size	Num. Of training patterns	Avg. of MAE	Avg. training time (Sec.)
HOSL-1	10-10	600	0.023098	9,042
HOSL-2	20-10	600	0.022937	9,068
HOSL-3	30-10	600	0.009271	9,354
HOSL-4	40-10	600	0.009274	9,867
HOSL-5	50-10	600	0.009272	9,915
HOML-1	10-12-10	600	0.003033	1,882
HOML-2	20-24-10	600	0.002926	1,268
HOML-3	30-36-10	600	0.002879	1,121
HOML-4	40-48-10	600	0.002865	1,043
HOML-5	50-60-10	600	0.002860	1,231

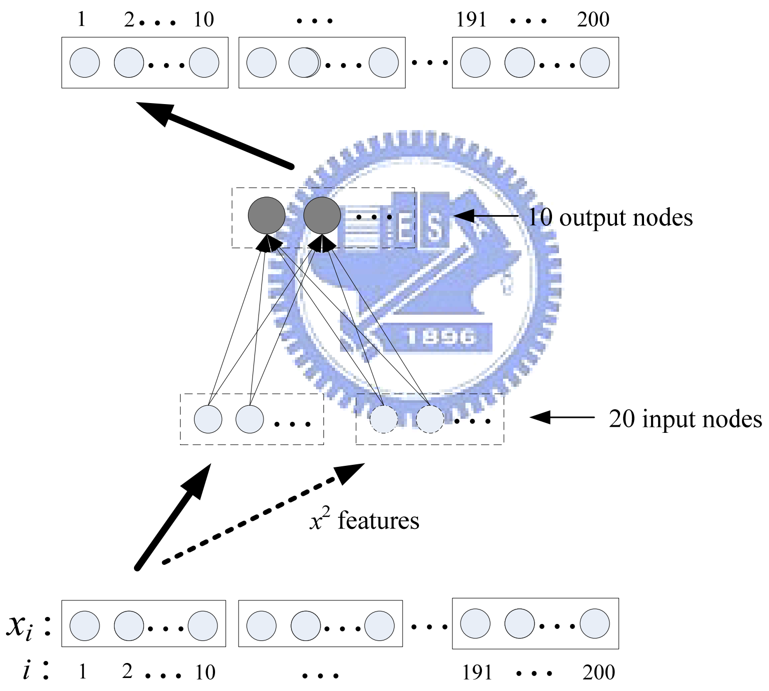


Fig. 4-9. HOSL-2 with original and expanded features.

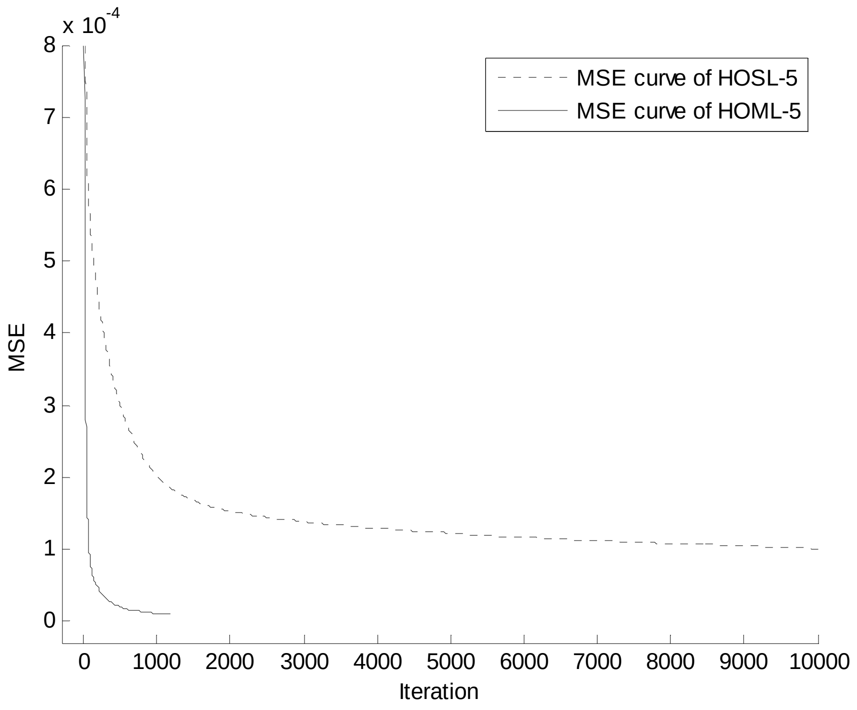


Fig. 4-10. Comparison of MSE curve between HOML-5 and HOSL-5.

B. Experiments of HOML with One Hidden Layer

We use the networks from HOML-1 to HOML-5 with one hidden layer. The learning rate is 0.6, the momentum parameter is 0.4, and the stopping error is 10^{-5}. 31 trials are done for average performance for each network type. The testing results are shown in Table 4-10, and the average of mean absolute error of each network is plotted in Figure 4-11. From the experimental results, we make the following two discoveries:

First, the HOML-1 that without expanding the input features yields the largest average of mean absolute error. The average of mean absolute error is smaller with adding higher-order features on the input features (from HOML-2 to HOML-5). It is obvious that the performance of HOML is better than general MLP network. Figure 4-12 is an example that shows the testing result of HOML-5 on dataset number 23.

Second, the average training time of HOML-2 to HOML-5 that with higher-order features is shorter. In 31 trials of each HOML network, we plot the training time for each testing dataset in Figure 4-13. In order to emphasize the training time for HOML-1, a solid line connects the nodes. In Figure 4-13, most of the training times of HOML-1 are longest among 5 network types. The networks with higher-order features have more input nodes than HOML-1; however, they have the shorter average training time (also shown in average training time column of Table 4-10). The reason is that the output of networks with higher-order features can fit the desired output more non-linear, the convergence becomes more quickly.

Table 4-10. Testing result of each type of HOML.

	Network size	Avg. of MAE	Smallest MAE	Avg. training time (sec.)
HOML-1	10-12-10	0.003008	0.002065	2,105
HOML-2	20-24-10	0.002514	0.002149	1,804
HOML-3	30-36-10	0.002496	0.002040	1,483
HOML-4	40-48-10	0.002501	0.002075	1,633
HOML-5	50-60-10	0.002407	0.001912	1,663

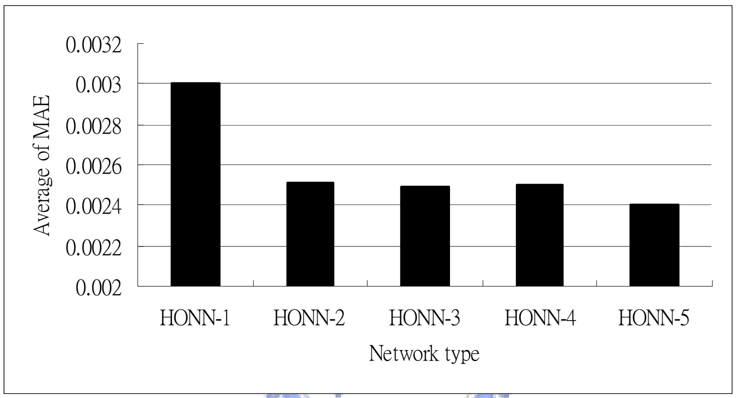


Fig. 4-11. Average of mean absolute error in different HOML networks.

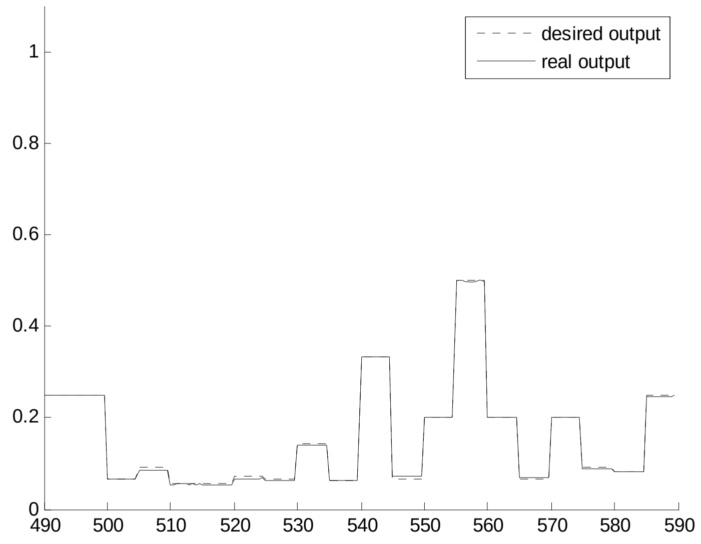


Fig. 4-12. Testing result of HOML-5 on dataset number 23.

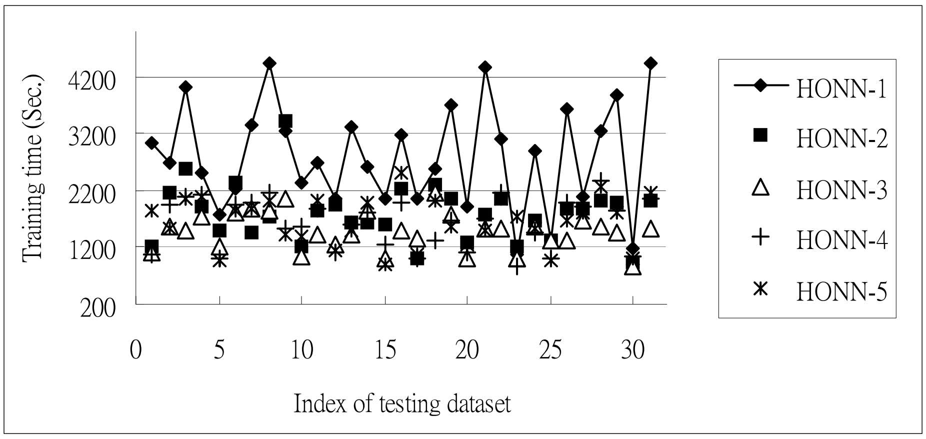


Fig. 4-13. Training time for each dataset in 5 HOML networks.

4.6 Higher-Order Feature Multi-Layer Neural Net with Conjugate Gradient (HOCG)

In this section, we use neural networks that are trained by conjugate gradient method instead of the gradient descent method, and also expand the input features using the higher-order functions which from second-order function to 5th-order function. We design 5 higher-order feature neural nets using these functions: HOCG-1, HOCG-2, HOCG-3, HOCG-4, and HOCG-5, and are listed in Table 4-11. These experiments are run with toolbox of Matlab ® v7.1 which use function newff(…) to create a neural network and assign the search routine as *Golden Section Search* (srchgol).

A. Evaluation of HOCG Networks

Table 4-12 shows the testing result of each network type. The stopping error is set to 10^{-5} for all experiments. Each network performance is evaluated by the average of mean absolute errors of 31 trials that achieve the training goal. From Table 4-10 and 4-12, the experimental results show that the neural networks trained by conjugate gradient method have better performance than those trained by gradient descent method. The average of mean absolute error of HOCG network is smaller and the average training time is shorter than corresponding HOML. Table 4-13 shows the comparison of average of mean absolute error in network performance between HOML and HOCG. Among all these 5 network types, HOCG-3 yields the largest improvement (16.1%) than the others.

Table 4-11. Five different HOCG types.

Network type	Features (include bias)	Network size
HOCG-1	$1+x$	10-12-10
HOCG-2	$1+x+x^2$	20-24-10
HOCG-3	$1+x+x^2+x^3$	30-36-10
HOCG-4	$1+x+x^2+x^3+x^4$	40-48-10
HOCG-5	$1+x+x^2+x^3+x^4+x^5$	50-60-10

Table 4-12. Comparison of each type of HOCG.

	Network size	Avg. of MAE	Smallest MAE	Avg. training time (sec.)
HOCG-1	10-12-10	0.002861	0.002326	558
HOCG-2	20-24-10	0.002475	0.002018	529
HOCG-3	30-36-10	0.002095	0.001824	616
HOCG-4	40-48-10	0.002369	0.001792	395
HOCG-5	50-60-10	0.002154	0.001765	408

Table 4-13. Comparison of average of mean absolute error between HOML and HOCG.

Network	Avg. of MAE	Network	Avg. of MAE	Improvement
HOML-1	0.003008	HOCG-1	0.002861	4.9%
HOML-2	0.002514	HOCG-2	0.002475	1.5%
HOML-3	0.002496	HOCG-3	0.002095	16.1%
HOML-4	0.002501	HOCG-4	0.002369	5.3%
HOML-5	0.002407	HOCG-5	0.002154	10.5%

B. Discussion of Divergence

The training process can be observed by monitoring the MSE curve. Here we summarize two of the reasons that may cause the learning unsuccessfully.

(1) The initial weight values.

Gradient-based method starts the learning procedure from initial connection weights, then update iteratively with the selected search direction and the step size. Since the error surface of neural network has numerous local minima, improper initial connection weights may cause the learning unsuccessfully. Figure 4-14 shows one of the testing cases in HOCG-5 that tests on dataset number 10. The error can not go down after large iterations. On the contrary, Figure 4-15 shows the result of another testing of HOCG-5 on the same testing dataset, and achieves the training goal at iteration 1,556 in this trial.

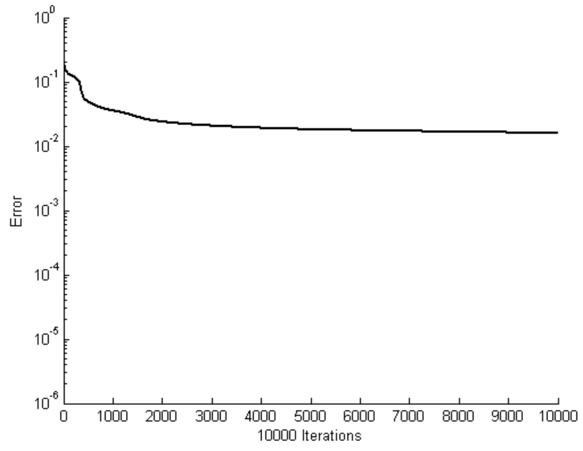


Fig. 4-14. Testing case of HOCG-5 on dataset number 10 that does not achieve the training goal.

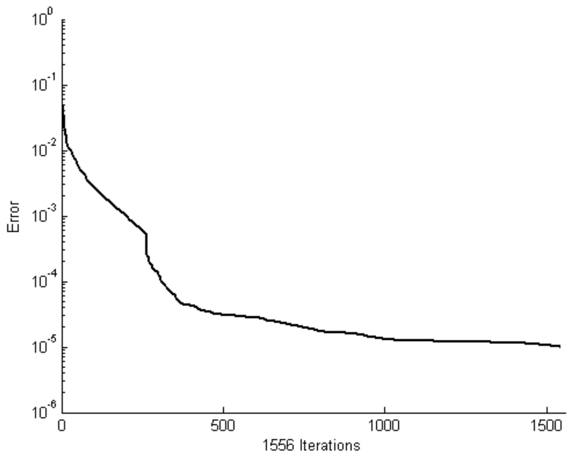


Fig. 4-15. Testing case of HOCG-5 on dataset number 10 that achieves the training goal.

(2) The maximal number of iterations.

The learning procedure may stop if the maximal number of iterations is not set to large enough. Figure 4-16 shows one of the test cases in HOCG-2 that tests on dataset number 12. The training converges at iteration 3,315, but no effective training is observed before about 2,600 iterations. Obviously, the training will fail to achieve the training goal if the maximal number of iterations is set to less than 2,600 in this trial.

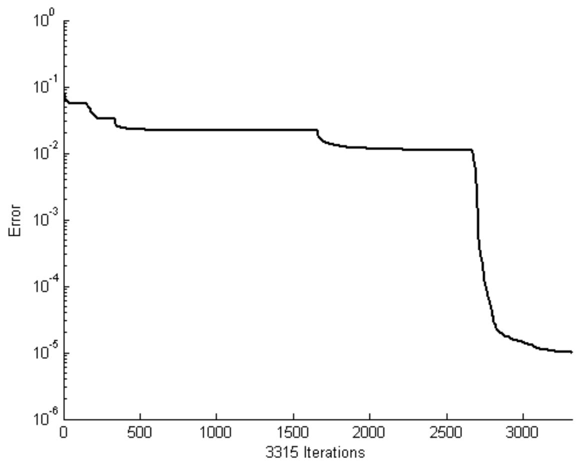


Fig. 4-16. Testing case of HOCG-2 on dataset number 12.

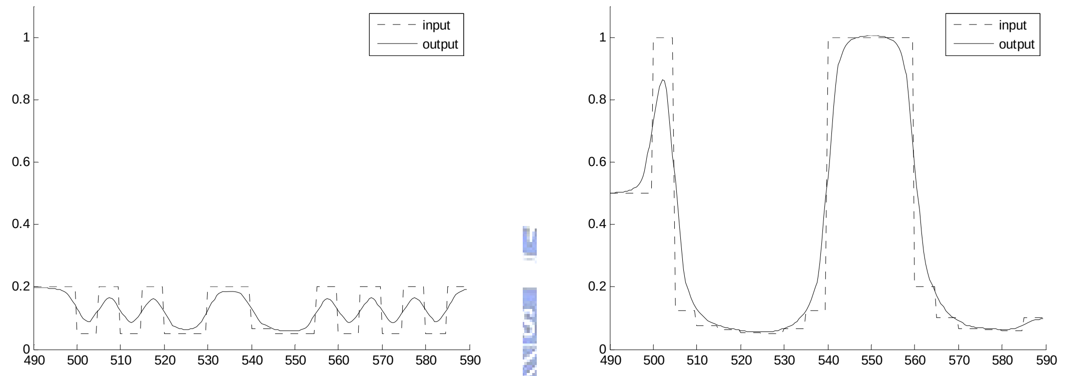
4.7 Experiments on Reversing the Input and Output of Well Logging Data

A. Determine the Suitable Number of Input Nodes of Network

In order to generate the synthetic training data from the desired true formation conductivity, we reverse the input and the output of well logging data. That means the original input becomes the desired output and the original desired output now becomes the input. By observing some datasets, such as number 2 and 5 as shown in Figure 4-17, some data intervals, for example, the depth from 490 to 500 feet in dataset number 2 and the depth from 540 to 560 feet in dataset number 5 have the same input features (the dotted line), so a network with short length 10 or 20 input nodes may not be suitable for training with these datasets. To find the number of input nodes, we compare the network performance with 10, 20, 40, and 50 input nodes. 10 experiments are done to get the average performance. The used network model is HOML-1 that uses the gradient descent method. The stopping error is set to 0.0005, the learning rate is 0.6, the momentum parameter is 0.4, and the used testing dataset is number 1, 5, 10, 12, 15, 17, 20, 23, 25, and 30. The number of training datasets is 30 for each testing. In Table 4-14, HOML-1 with 10 and 20 input nodes can not converged. In this case, HOML-1 is not able to achieve the training goal. However, it converged when the number of input nodes is 40 and 50, and the one with 50 input nodes has better performance, so we use the HOML networks with 50 inputs nodes to do experiments of reversing the input and output of well logging data.

Table 4-14. Average of mean absolute error with 10, 20, 40, and 50 inputs nodes in HOML-1.

Number of input nodes	Network size	Avg. of MAE
10	10-12-10	Diverged
20	20-24-20	Diverged
40	40-48-40	0.02398
50	50-60-50	0.02143



(a) Dataset number 2. (b) Dataset number 5.

Fig. 4-17. Two well logging datasets.

B. Experimental Results of HOML and HOCG

From Table 4-13, the HOML-3, 4, 5 and the HOCG-3, 4, 5 have smaller average of mean absolute error, so we use these three models with 50 input and output nodes to do the experiments on reversing the input and output of well logging data. The number of hidden nodes is 1.2 times of the number of input nodes. The learning rate for HOML is 0.6 and the momentum parameter is 0.4. The stopping error is set to 0.0005. Table 4-15 and 4-16 show the comparison of average of mean absolute error and training time of HOML and HOCG respectively.

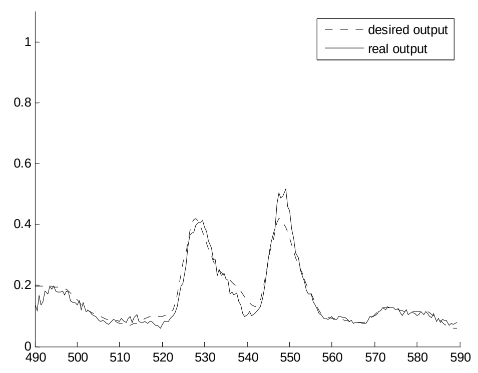
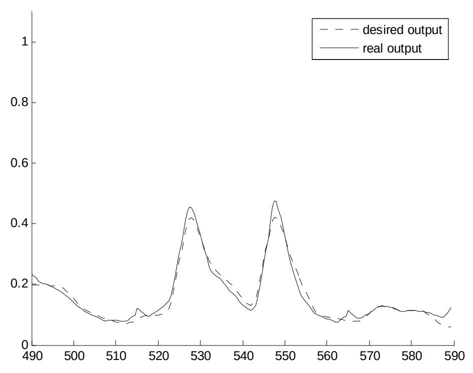
The testing of HOML-3 on dataset number 10, HOML-4 on dataset number 20, and HOML-5 on dataset number 30 are shown in Figure 4-18 respectively. From the testing result of HOML-4 and HOCG-4 in Figure 4-18(b), HOML that trained by gradient descent method has the better performance than HOCG that trained by conjugate gradient method. However, as shown in Table 4-15, among all three types we used, the average of mean absolute error of HOCG networks are smaller than HOML network types. From Table 4-16, for larger-size network (HOML-5 and HOCG-5), they do not have much difference in the average training time although the difference of average of mean absolute error between them is significant. On the other hand, for smaller-size network, they have much more difference in average training time as HOML-3 and HOCG-3. For small network, conjugate gradient method has more significant improvement in training time to convergence.

Table 4-15. Average of mean absolute error of HOML and HOCG.

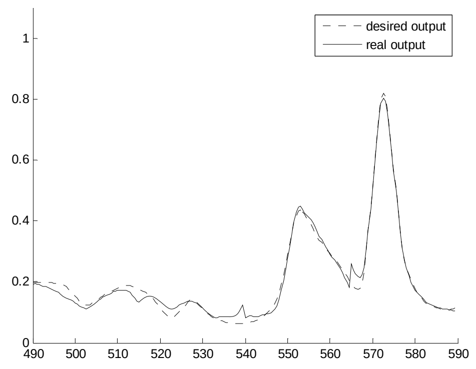
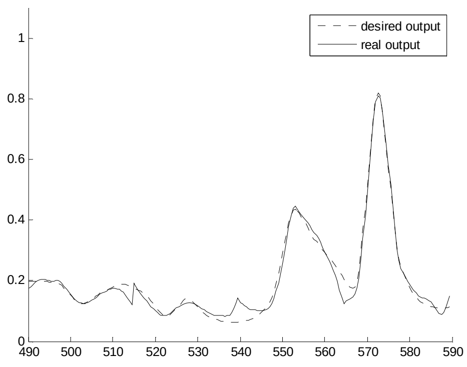
	HOML-3 vs. HOCG-3	HOML-4 vs. HOCG-4	HOML-5 vs. HOCG-5
Network size	150-180-50	200-240-50	250-300-50
HOML	0.01684	0.01965	0.01947
HOCG	0.01507	0.01758	0.01361

Table 4-16. Training time of HOML and HOCG in seconds.

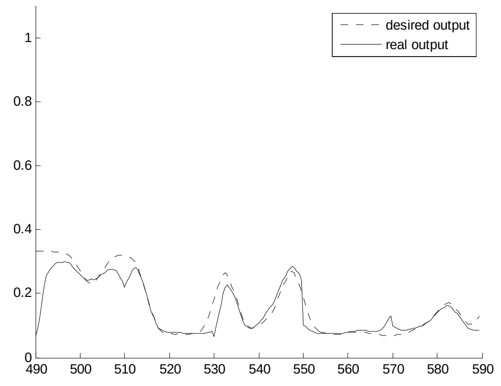
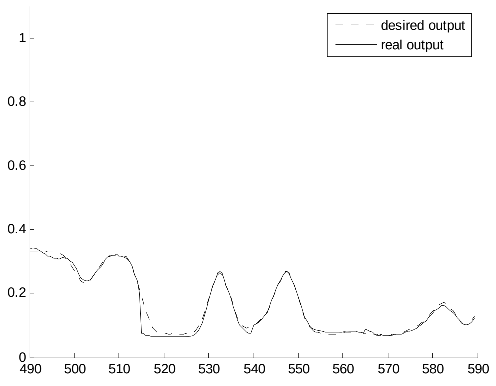
	HOML-3 vs. HOCG-3	HOML-4 vs. HOCG-4	HOML-5 vs. HOCG-5
Network size	150-180-50	200-240-50	250-300-50
HOML	3,957	3,864	5,284
HOCG	1,768	2,239	4,729

HOML-3	HOCG-3
	
On dataset number 10, MAE is 0.01817.	On dataset number 10, MAE is 0.01481.

(a) HOML-3 vs. HOCG-3.

HOML-4	HOCG-4
	
On dataset number 20, MAE is 0.01356.	On dataset number 20, MAE is 0.01569.

(b) HOML-4 vs. HOCG-4.

HOML-5	HOCG-5
	
On dataset number 30, MAE is 0.02326.	On dataset number 30, MAE is 0.00992.

(c) HOML-5 vs. HOCG-5.

Fig. 4-18. Comparison of network performance of HOML and HOCG.

4.8 Summary

From Table 4-10, 4-12 and 4-15, we summarize the testing performance of networks and shown in Table 4-17 and Table 4-18. In Table 4-17, each HOCG network has smaller average of mean absolute error and shorter average training time than HOML network, which shows that the conjugate gradient method improves the training efficiency. Also the network with higher-order features has better performance than general MLP network that using both gradient descent method and conjugate gradient method, which shows that the proposed higher-order feature neural nets are more suitable for well logging data inversion. The same result can be seen in Table 4-18, which shows the comparison of experimental results of reversing the input and output of well logging data. Moreover, the average training time of experiments of reversing the input and output of well logging data is much more than previous normal inverse procedure. One of the reasons is that the network size is larger because we use 50 input nodes for each higher-order feature neural net. Another possible reason is that the input features are distributed in the shape of straight lines as shown in Figure 4-17.

Table 4-17. Comparison of HOML and HOCG in average of mean absolute error and average training time.

	Network size	Avg. of MAE	Avg. training time (Sec.)
HOML-1	10-12-10	0.003008	2,105
HOCG-1	10-12-10	0.002861	558
HOML-2	20-24-10	0.002514	1,804
HOCG-2	20-24-10	0.002475	529
HOML-3	30-36-10	0.002496	1,483
HOCG-3	30-36-10	0.002095	616
HOML-4	40-48-10	0.002501	1,633
HOCG-4	40-48-10	0.002369	395
HOML-5	50-60-10	0.002407	1,663
HOCG-5	50-60-10	0.002154	408

Table 4-18. Comparison of experimental results of reversing the input and output of well logging data.

	Network size	Avg. of MAE	Avg. training time (Sec.)
HOML-3	150-180-50	0.01684	3,957
HOCG-3	150-180-50	0.01507	1,768
HOML-4	200-240-50	0.01965	3,864
HOCG-4	200-240-50	0.01758	2,239
HOML-5	250-300-50	0.01947	5,284
HOCG-5	250-300-50	0.01361	4,729

5. Genetic Algorithm (GA) on Well Logging Inversion

5.1 Introduction

Genetic algorithm (GA) was investigated by John Holland et al. in 1975 [17]. It was based on the biological evolution. GA's learning is a competition among all chromosomes in population, and treats these chromosomes for the potential solutions to the target problem. Population consists of a set of chromosomes and a chromosome consists of a number of genes. The number of individuals in population is called the population size A chromosome is also called an individual in GA, and an iteration referred to the GA is called a generation. In each generation, individuals are operated by genetic algorithm procedure to form the next generation. The fitness value can be obtained through fitness function and is used to evaluate each individual. Fitness value can be regarded as a measure of indicating how good an individual is. The fitness function is specifically designed for the different problem. Through competition the individual with better fitness value has higher probability to be reproduced and generates offspring, in such a way that the solution will be better and better to the target problem. GA is a global search algorithm that provides chance to avoid getting trapped at local minimum, and then often treated as one of the powerful tool to optimization problem. Moreover, GA needs only a fitness function and does not use the gradient information. Because it is easy to implement, GA is widely used in many applications. [18]-[19].

The GA process can be summarized as following steps:

(1) Generate the initial population with many individuals.

The GA process starts with an initial population of random chromosomes. A chromosome consists of one or more genes. The basic way of coding is in binary format. Figure 5-1 shows a chromosome consists of 3 genes and each gene consist of 3 bits.

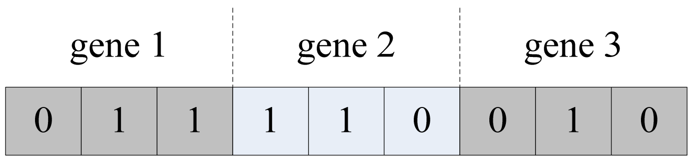


Fig. 5-1. Binary format of an individual with 3 genes.

(2) Evaluate the fitness value of each individual.

Each chromosome will then be evaluated by computing its fitness value. Individuals with the higher fitness value have more chance to generate offspring during evolution because through competition they have more chance to be reproduced in reproduction phase.

(3) Termination condition.

Rank the fitness values and find the best individual. The algorithm is terminated when the best individual meets the desired requirement or the evolution reaches the maximal number of generations.

(4) Reproduction.

Reproduction is the process for selecting individuals to be reproduced for the next generation. The methods of roulette wheel selection and tournament selection can be used to select individuals.

1. Roulette wheel selection.

   In roulette wheel selection method, the population is viewed as mapping on a roulette wheel. Each individual is represented by a space that proportionally corresponds to its fitness value, which means, an individual with higher fitness value occupies more space in the roulette wheel so that has more chance to be reproduced. For example, the number of individuals of population is N, f_i is the fitness value of individual i, and then the number of individual i to be reproduced is:

$$N * (f_i / \sum_{j=1}^{N} f_j)$$

2. Tournament selection.

   There are three stages in this reproduction method:

   (a) Select two or more individuals.

   (b) Reproduce the individual that has the highest fitness value among them.

   (c) Repeat step (a) and step (b) until the number of reproduced individuals is equal to the current population size.

(5) Crossover.

Crossover is the process of exchanging gene information of randomly selected two individuals by the crossover probability P_r, one basic crossover process is the single point crossover as illustrated in Figure 5-2.

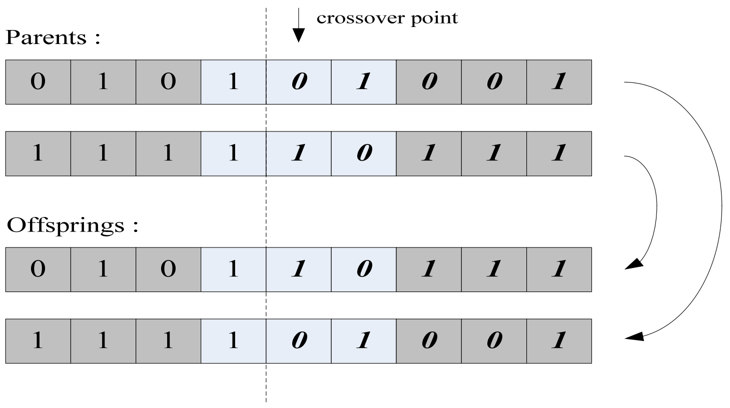


Fig. 5-2. Single point crossover from two individuals, each element of individual is a gene.

(6) Mutation.

Mutation is the process to flip a bit in an individual. The mutation probability P_m decides how many bits will be mutated in population. For example, set mutation probability of 1% means totally 1% bits in whole population will be flipped.

Reproduction, crossover, and mutation are three major GA operators that used to produce the new population. A new generation starts by going to step (2). The flowchart of genetic algorithm is shown in Figure 5-3.

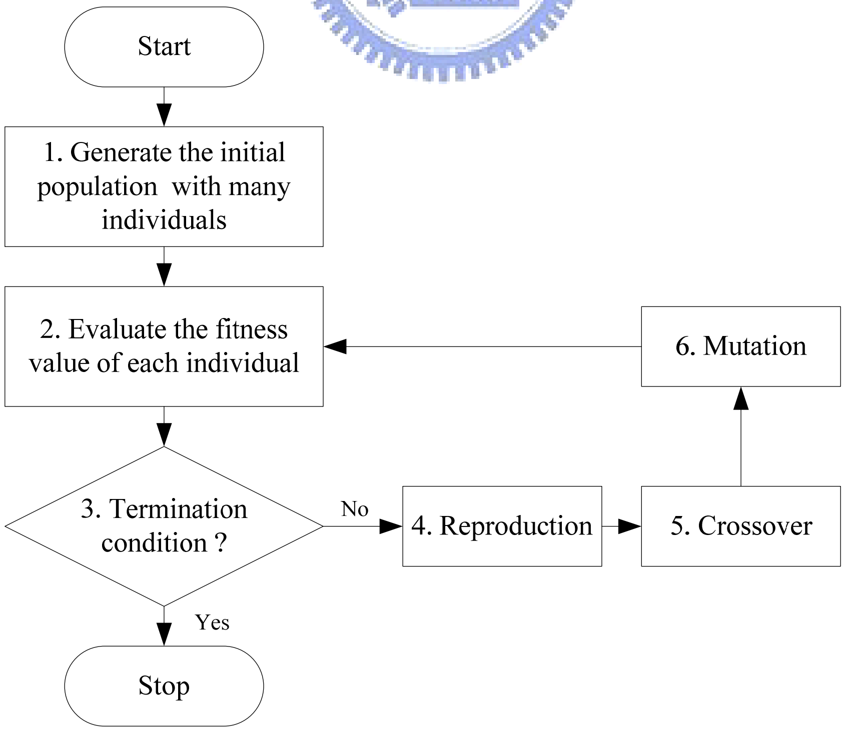

Start
1. Generate the initial population with many individuals
2. Evaluate the fitness value of each individual
3. Termination condition ?
No
Yes
4. Reproduction
5. Crossover
6. Mutation
Stop

Fig. 5-3. Flowchart of genetic algorithm.

5.2 Real-Coded Genetic Algorithm (RCGA)

The difference between the RCGA and GA is the coding format of individuals [20]. In RCGA, the individuals are directly represented with real number instead of the binary string. The binary format has the disadvantage in computational cost because of encode and decode phases. In addition, RCGA deals with continuous search space without sacrificing numerical precision as the binary format does. RCGA applies different crossover operator and mutation operator from GA. In order to do crossover and mutation on real-coded individuals, Michalewicz [21] introduced the arithmetic crossover and arithmetic mutation.

(1) Arithmetic crossover.

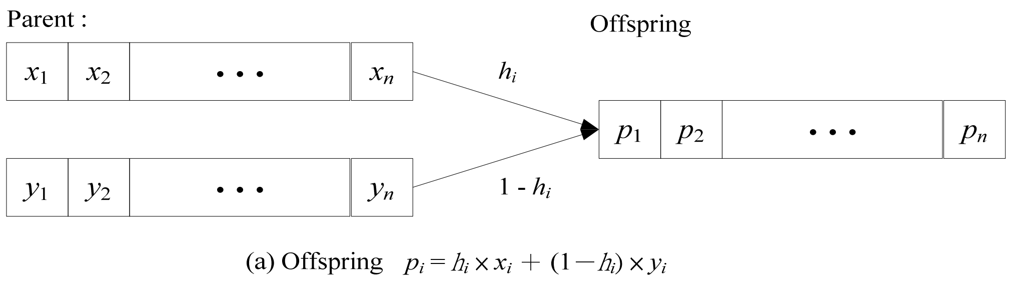
Consider two selected individuals $\mathbf{x} = \{x_1, \ldots, x_n\}$ and $\mathbf{y} = \{y_1, \ldots, y_n\}$ to produce offspring **p** and **q** as shown in Figure 5-4:

$$p_i = h_i \times x_i + (1-h_i) \times y_i$$
$$q_i = (1-h_i) \times x_i + h_i \times y_i$$

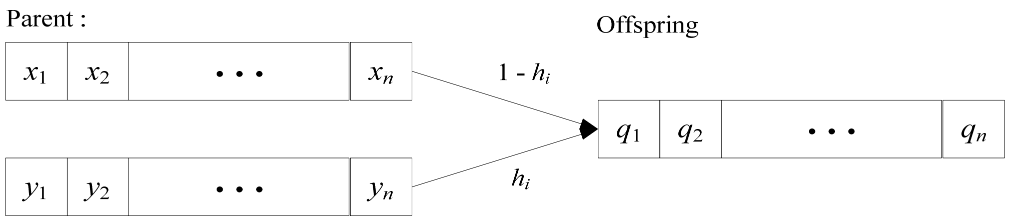
where h_i is a uniform random crossover parameter.

(2) Arithmetic mutation.

Given an individual $\mathbf{x} = (x_1, x_2, \ldots, x_i, \ldots, x_n)$ that is selected to be mutated and an interval $[a_i, b_i]$ where a_i and b_i are the lower and upper bound for gene x_i. While doing mutation, a new gene x_i' is uniform randomly selected from range $[a_i, b_i]$ to replace the original gene x_i and then a new individual $\mathbf{x'} = (x_1, x_2, \ldots, x_i', \ldots, x_n)$ is formed.



(a) Offspring $p_i = h_i \times x_i + (1-h_i) \times y_i$



(b) Offspring $q_i = (1-h_i) \times x_i + h_i \times y_i$

Fig. 5-4. Arithmetic crossover.

5.3 Experiments on Well Logging Data Inversion

We use the hybrid method that combines neural network and GA to do the well logging data inversion. In section 5.3.1, we describe how an individual can be used to represent a neural network. In section 5.3.2, we describe the reason and give flowchart of the combination of neural network and GA. In section 5.3.3, we do experiments to evaluate the performance of the hybrid method. In section 5.3.4, we do experiments on reversing the input and output of well logging data, and section 5.3.5 is the summary of experiments.

5.3.1. Representation of Weights of Neural Network

When applying GA in finding the weight values of neural network, each gene of individual represents a connection weight of neural network and each individual forms a neural network. The number of gene of an individual is equal to the total number of connection weight of a network. For example, there are I input nodes, J hidden nodes, and K output nodes in a neural network, an individual that represents the network have L genes, where $L = ((I+1)(J)+(J+1)(K))$. Figure 5-5 shows how an individual can be used to represent a network. The network has 2 input nodes, 3 hidden nodes, and 2 output nodes, the total number of connection weight is (2+1)(3)+(3+1)(2) = 17. Since each gene represents a connection weight, we can use an individual that contains 17 genes to represent such a MLP network.

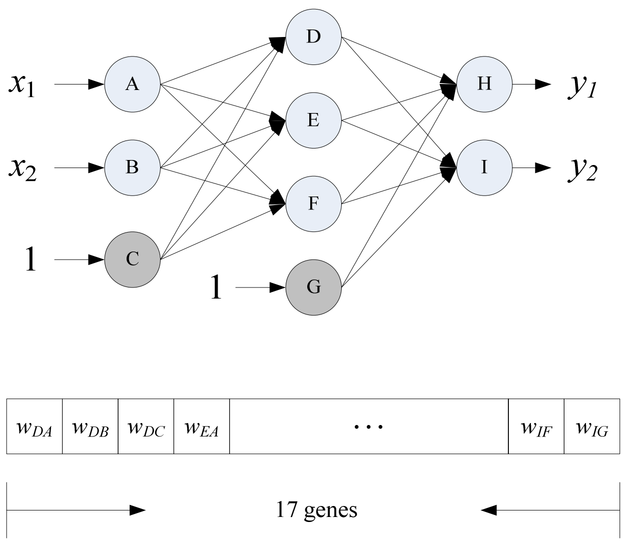


Fig. 5-5. A MLP network represented by an individual.

5.3.2. Combination of Neural Network and GA

GA offers a way to search for connection weights of MLP networks. However, GA is relatively slow in local search although it is good at global search. The learning efficiency of GA can be improved by integrating a local search procedure into the evolution. The local search algorithm could be BP learning algorithm [22]. In our experiment, GA is the main procedure that used to find the connection weights of MLP network, and in order to improve the learning efficiency of GA, a BP training using gradient descent is incorporated for N_b iterations with the best individual every N_g GA generations. The fitness function is defined as the inverse of mean squared error. The algorithm is described as follows, and the flowchart is shown in Figure 5-6.

Algorithm 5-1: Hybrid method for well logging data inversion.

Input: Well logging datasets and the corresponding desired output values. 30 datasets are used in training and 1 dataset is used in testing.

Output: Mean absolute error using 1 testing dataset.

```
Step 1: Initialization.
        1. Set the stopping error.
        2. Set parameters of GA and BP training.
        3. Generate the initial population with many individuals, and each individual
           forms a network.
Step 2: Input 30 well logging datasets to each individual for training, and do genetic
        algorithm for one generation.
Step 3: Compute the fitness values of all individuals.
Step 4: Find the best one individual to do BP training that use gradient descent
        method for $N_b$ iterations.
Step 5: Compute the fitness value of individual that operated in step 4.
        If the fitness value of best one individual is less than the stopping error or
        the computation reaches the maximal number of generations, then algorithm
        stops.
        Otherwise, repeat by going to step 2.
Step 6: Input 1 testing well logging dataset, and compute the mean absolute error.
```

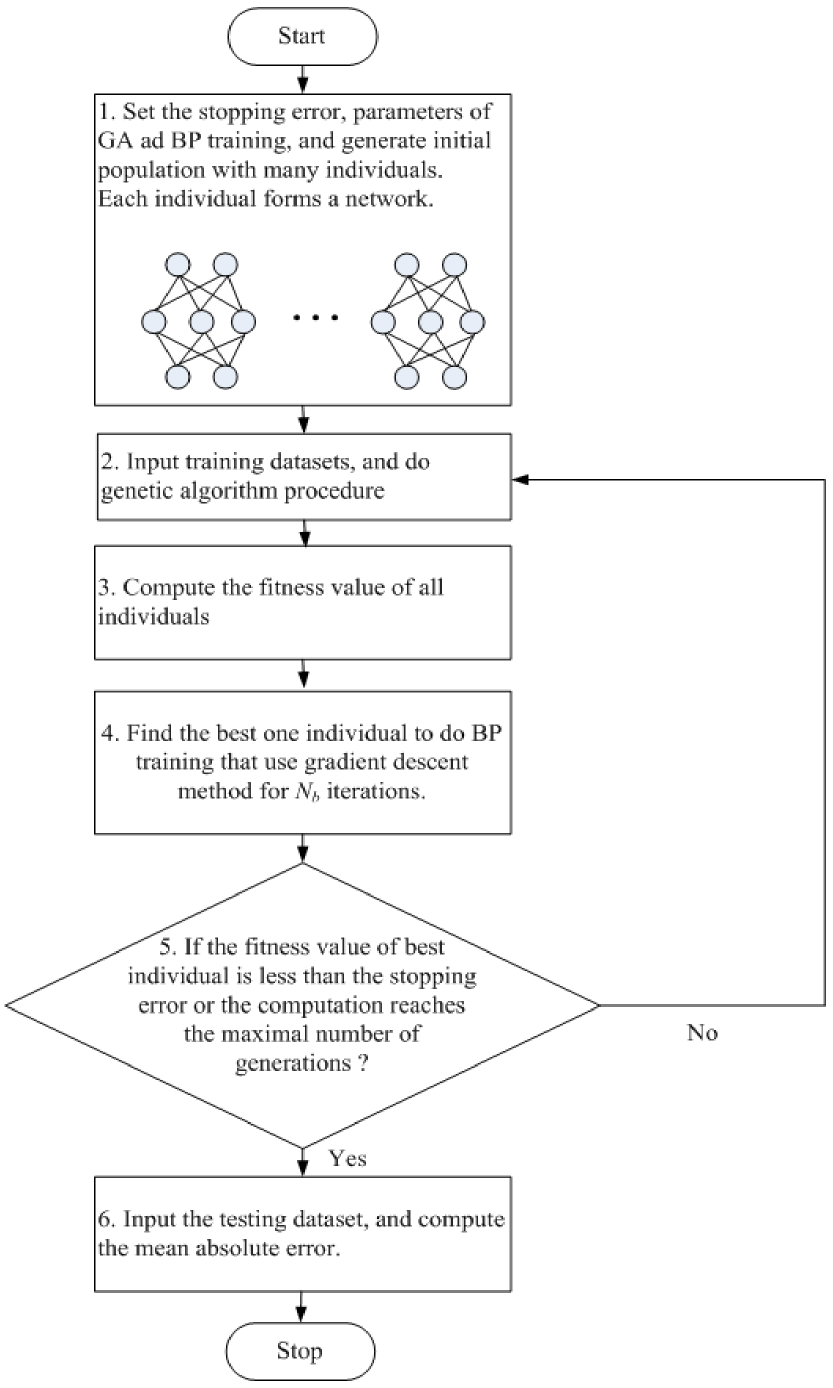

Start
1. Set the stopping error, parameters of GA ad BP training, and generate initial population with many individuals. Each individual forms a network.
2. Input training datasets, and do genetic algorithm procedure
3. Compute the fitness value of all individuals
4. Find the best one individual to do BP training that use gradient descent method for N_b iterations.
5. If the fitness value of best individual is less than the stopping error or the computation reaches the maximal number of generations ?
No
Yes
6. Input the testing dataset, and compute the mean absolute error.
Stop

Fig. 5-6. Flowchart of the hybrid method on well logging inversion.

5.3.3 Evaluation of Hybrid Neural Network and GA

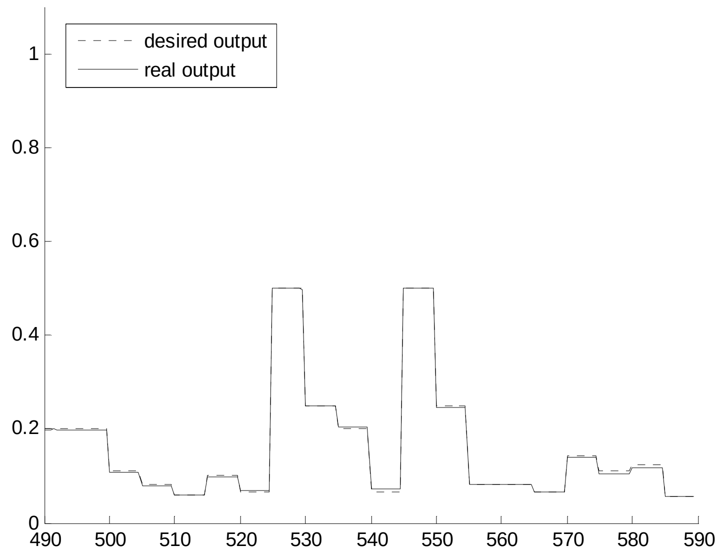
Global search procedure such as GA is usually computationally expensive, and it would be better to employ local search procedure, so we do BP training with the best individual after each GA generation. The local search procedure here is used to refine the network with the best individual, not to find the final optimal weight with that individual, so the number of iterations is not necessary to be large, and we set

N_b = 10. The number of individual in population is 30, and the maximal number of generations is 10,000. The crossover rate and the mutation rate are set to 0.8 and 0.001 respectively. The crossover parameter is randomly selected in range [0, 1], and the mutation range is set to range [-1.0, 1.0]. The used reproduction method is tournament selection. We use hybrid method of HOML-1 with GA to HOML-5 with GA, the learning rate in HOML networks is 0.6 and the momentum parameter is 0.4. The stopping error is set to 10^{-5}. 10 experiments are done for each hybrid method to get the average performance. The used testing datasets are 1, 5, 10, 12, 15, 17, 20, 23, 25, and 30 respectively, and the number of training datasets is 30. The testing result is shown in Table 5-1.

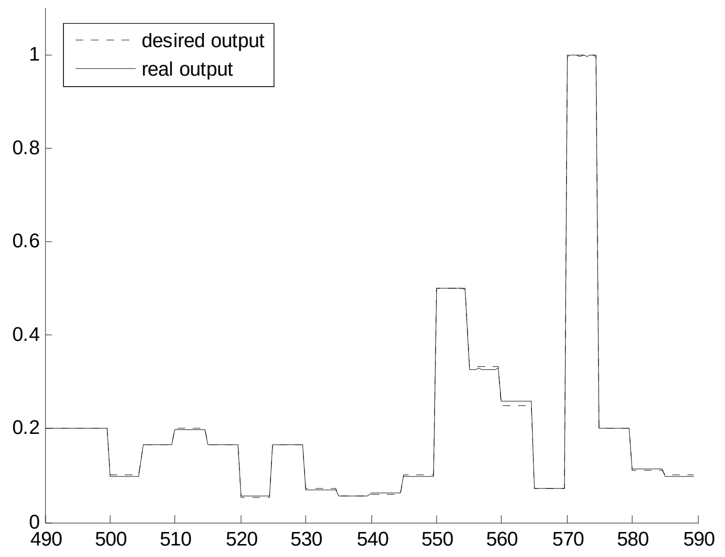
Table 5-1 shows the testing result of different architecture of hybrid methods. In our experiments, adding more higher-order terms in input features do not provide more improvement in reducing the mean absolute error. Among these 5 hybrid methods, HOML-3 with GA yields the smallest average of mean absolute error. By comparing Table 5-1 and the testing results of HOML as shown in Table 4-10, no significant difference about average of mean absolute error can be found. However, the average training time of hybrid method is much more than that of HOML. Figure 5-7 shows two of test cases. Figure 5-7(a) shows the test case of HOML-2 with GA on dataset number 10, and it is the test case that has the smallest mean absolute error among 10 trials. Figure 5-7(b) shows the test case of HOML-3 with GA on dataset number 20, and it is the test case that has the smallest mean absolute error among 10 trials.

Table 5-1. Testing results of HOML with GA.

	Network size	Num. of training patterns	Avg. of MAE	Smallest MAE	Avg. training time (sec.)
HOML-1 with GA	10-12-10	600	0.002927	0.002768	9,568
HOML-2 with GA	20-24-10	600	0.002492	0.002207	9,218
HOML-3 with GA	30-36-10	600	0.002349	0.002008	13,521
HOML-4 with GA	40-48-10	600	0.002729	0.002461	14,254
HOML-5 with GA	50-60-10	600	0.002658	0.002125	14,168



(a) Testing dataset on number 10 with the mean absolute error 0.002507 using HOML-2 with GA.



(b) Testing dataset on number 20 with the mean absolute error 0.002208 using HOML-3 with GA.

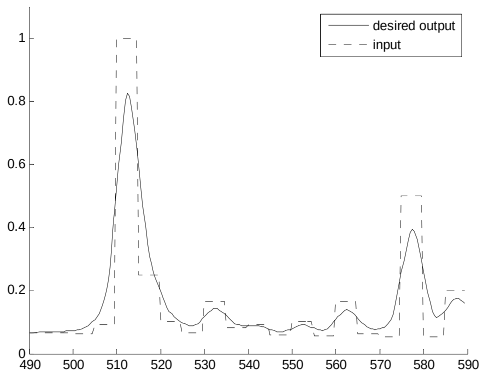
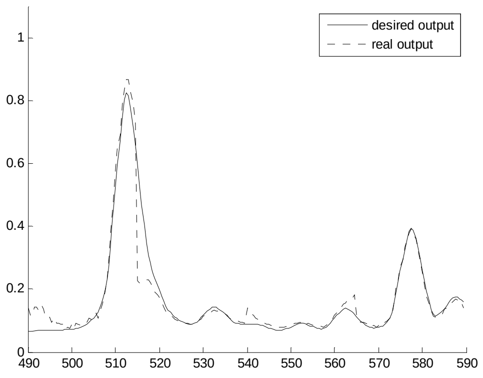
Fig. 5-7. Testing results in HOML-2 with GA and HOML-3 with GA.

5.3.4 Experiments on Reversing the Input and Output of Well Logging Data

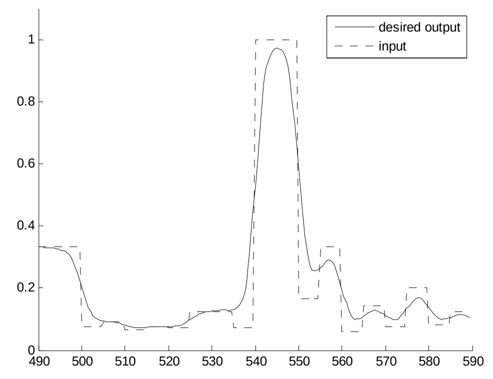
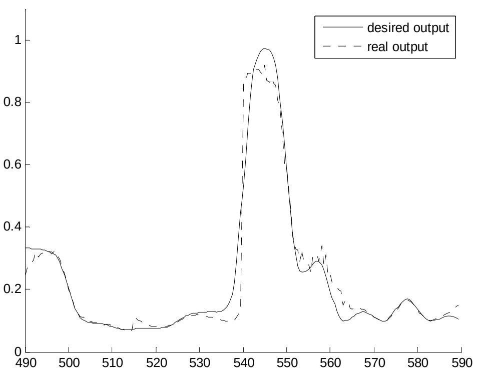
We use HOML-3 with GA to HOML-5 with GA to do the experiments on reversing the input and output of well logging data. The number of hidden nodes is 1.2 times of the number of input nodes. The learning rate for BP training is 0.6, the momentum parameter is 0.4, and the stopping error is set to 0.0005. 10 trials are done for experiment of each network type with dataset number 1, 5, 10, 12, 15, 17, 20, 23, 25, and 30 as the testing dataset. The number of training datasets is 30. The testing results are shown in Table 5-2. Figure 5-8 shows the tests of HOML-3 with GA to HOML-5 with GA on testing dataset number 15, 25, and 30 respectively. According to the testing results in Table 5-2, the average of mean absolute error in HOML-5 with GA is small. There is no significant difference about the average of mean absolute error in HOML-3 with GA and HOML-4 with GA.

Table 5-2. Average performance of three hybrid methods.

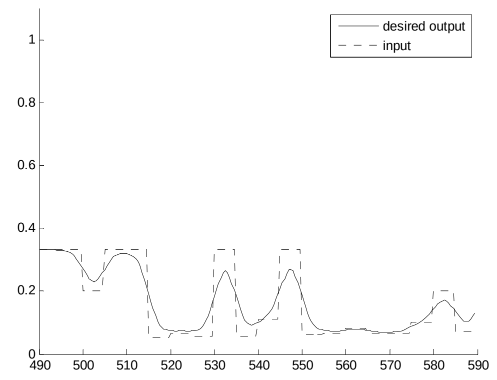
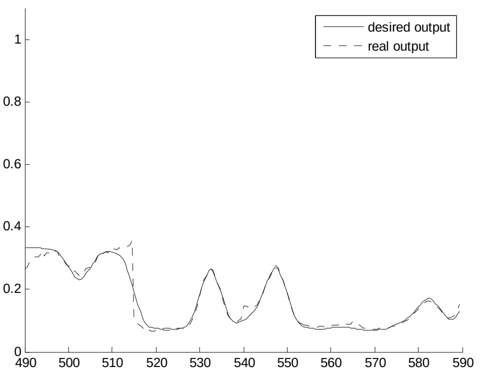
	Network size	Num. of training patterns	Avg. of MAE	Smallest MAE	Avg. training time (sec.)
HOML-3 with GA	150-180-50	120	0.02237	0.01861	7,628
HOML-4 with GA	200-240-50	120	0.02452	0.01749	9,205
HOML-5 with GA	250-300-50	120	0.01364	0.00983	11,458

Original dataset	Experimental result
	
	On dataset 15, MAE is 0.021522.

(a) Testing result on dataset number 15 using HOML-3 with GA.

Original dataset	Experimental result
	
	On dataset 25, MAE is 0.027113.

(b) Testing result on dataset number 25 using HOML-4 with GA.

Original dataset	Experimental result
	
	On dataset 30, MAE is 0.011405.

(c) Testing result on dataset number 30 using HOML-5 with GA.

Fig. 5-8. Testing results of HOML with GA.

5.3.5 Summary

Table 5-3 and Table 5-4 are summaries of the testing performance in Chapter 4 and Chapter 5. Table 5-3 shows the average of mean absolute error and the average training time of HOML-1 to HOML-5, HOCG-1 to HOCG-5, and HOML-1 with GA to HOML-5 with GA. From our testing results we make two conclusions. First, hybrid method that combines higher-order feature neural network and GA is not always converging to a better solution than higher-order feature neural network does. In HOML-4 with GA and HOML-5 with GA, the average of mean absolute error is slightly larger than that of HOML network. Second, as compared with gradient-based optimization technique that used by HOML and HOCG, GA requires much more training time to accomplish the training goal.

Table 5-4 shows the experimental results of reversing the input and output of well logging data of HOML-3 to HOML-5, HOCG-3 to HOCG-5, and HOML-3 with GA to HOML-5 with GA. In comparing the average of mean absolute error, HOCG-5 is the smallest one. And for average training time, HOCG-3 is the shortest one, which shows the HOCG network has best performance among all of these experiments.

Table 5-3. Comparison of HOML and HOCG in average of mean absolute error and average training time.

Network	Avg. of MAE	Avg. training time (Sec.)
HOML-1	0.003008	2,105
HOCG-1	0.002861	558
HOML-1 with GA	0.002927	9,568
HOML-2	0.002514	1,804
HOCG-2	0.002475	529
HOML-2 with GA	0.002492	9,218
HOML-3	0.002496	1,483
HOCG-3	0.002095	616
HOML-3 with GA	0.002349	13,521
HOML-4	0.002501	1,633
HOCG-4	0.002369	395
HOML-4 with GA	0.002729	14,254
HOML-5	0.002407	1,663
HOCG-5	0.002154	408
HOML-5 with GA	0.002658	14,168

Table 5-4. Comparison of experimental results of reversing the input and output of well logging data.

Network	Avg. of MAE	Avg. training time (Sec.)
HOML-3	0.01684	3,957
HOCG-3	0.01507	1,768
HOML-3 with GA	0.02237	7,628
HOML-4	0.01965	3,864
HOCG-4	0.01758	2,239
HOML-4 with GA	0.02452	9,205
HOML-5	0.01947	5,284
HOCG-5	0.01361	4,729
HOML-5 with GA	0.01364	11,458

6. Experiments on Real Field Logs

In order to verify the effectiveness of the well logging data inversion using our proposed higher-order feature neural nets, an inversion of real field logs are presented. In our study, the used real field log covers depth that from 5,577.5 to 6,682 feet. There are 2 points per foot so we have total 2,210 points in field logs.

The experimental results that listed in Table 5-3 show that the network type HOCG-3 has the best inverse performance, the mean absolute error can be down to about 0.209%, so we use HOCG-3 to invert the real field logs. The number of input nodes is 10 (not include bias) and the number of output nodes is 10. Using second-order and third-order transformation function expands input features, so the number of hidden nodes is 36. The maximal number of iterations is 10,000, and the stopping error is set to 10^{-5}. In each trial, we use 31 synthetic combined well logging datasets to training the network. After training, real field logs are transformed to conductivity and then presented to the trained network to get the inverted conductivity. The conductivity is then converted back to the true formation resistivity. In order to lower the impact of initial weights on final output, we test 20 trials and compute the average inverted conductivity. Figure 6-1 shows two samples of MSE curve of these 20 trials while in training. The average training time is 469 seconds and the inverted result is shown from Figure 6-2 to Figure 6-6 that represents an input/output mapping. The input is the field logs and the output is the inverted true formation resistivity. To view the result more detail, we split the inverted true formation resistivity into 4 sections and list in Figure 6-2 to Figure 6-5, and Figure 6-6 is the result of the whole field logs. The experimental result shows that our proposed higher-order feature neural net is an effective alternative to process the well logging data inversion.

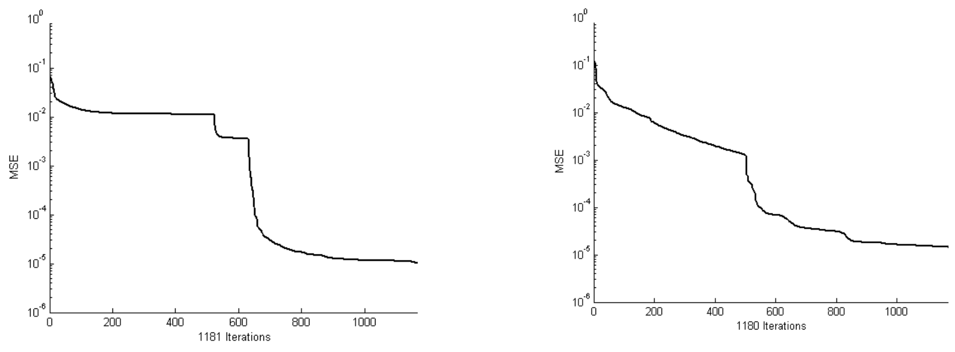


Fig. 6-1. Two samples of MSE curve while in training.

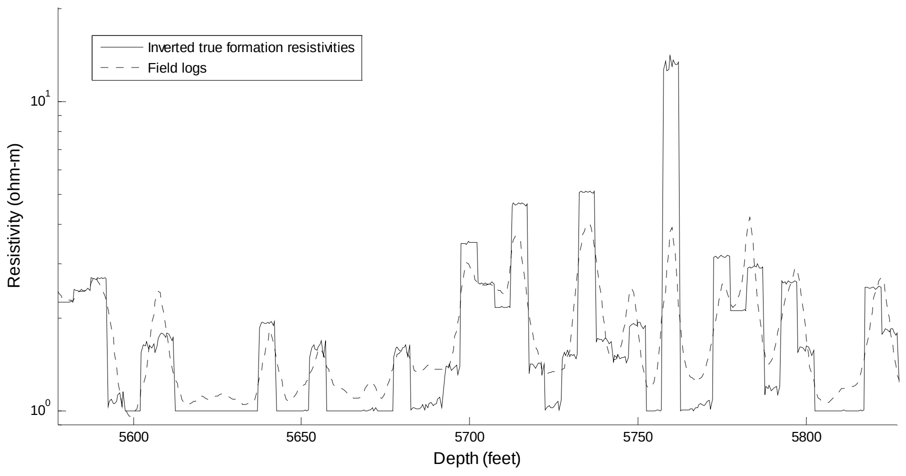


Fig. 6-2. The inverted true formation resistivity between 5,577.5 to 5,827 feet.

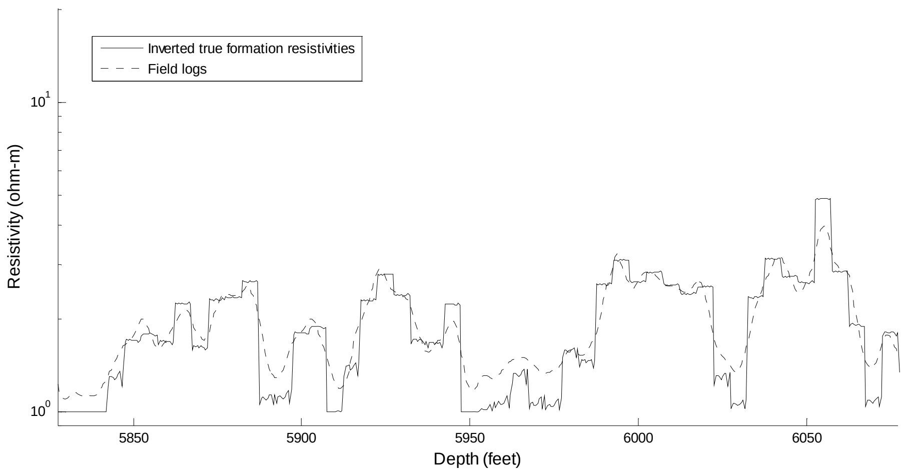


Fig. 6-3. The inverted true formation resistivity between 5,827.5 to 6,077 feet.

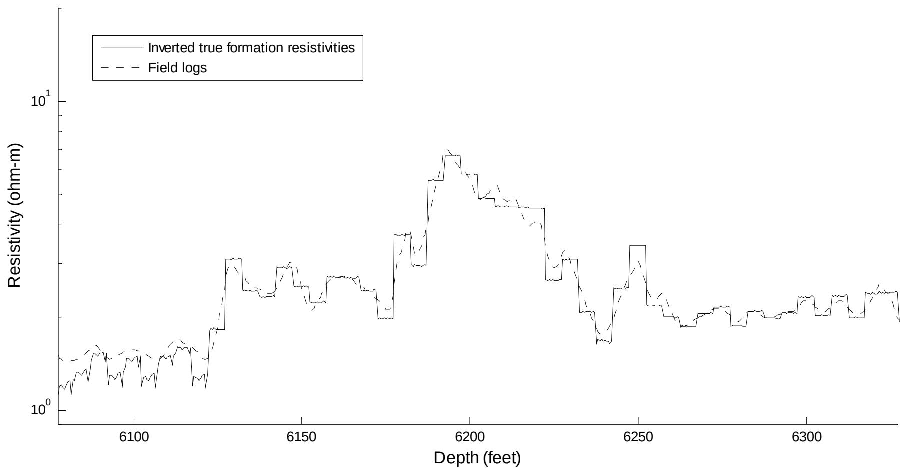


Fig. 6-4. The inverted true formation resistivity between 6,077.5 to 6,327 feet.

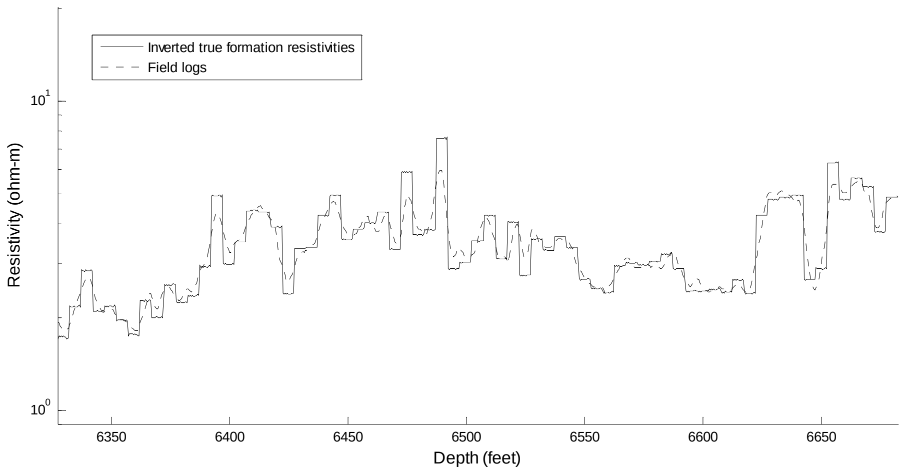
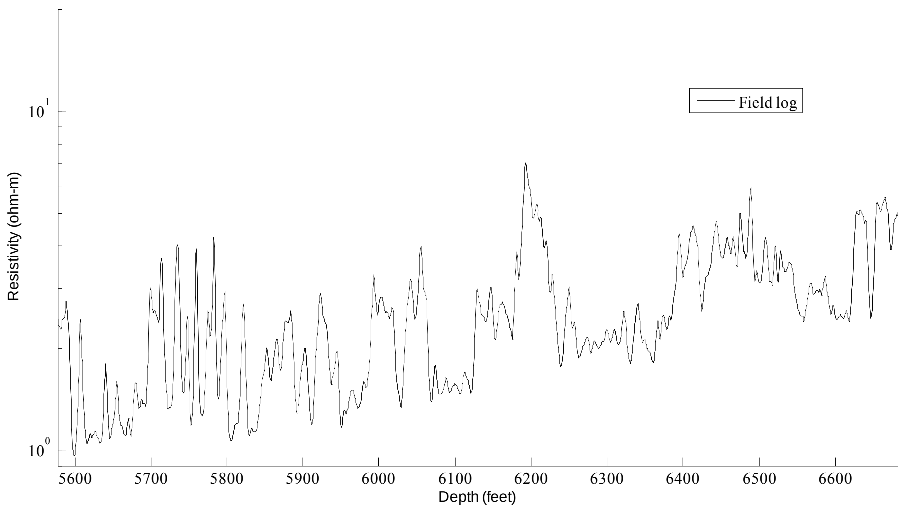
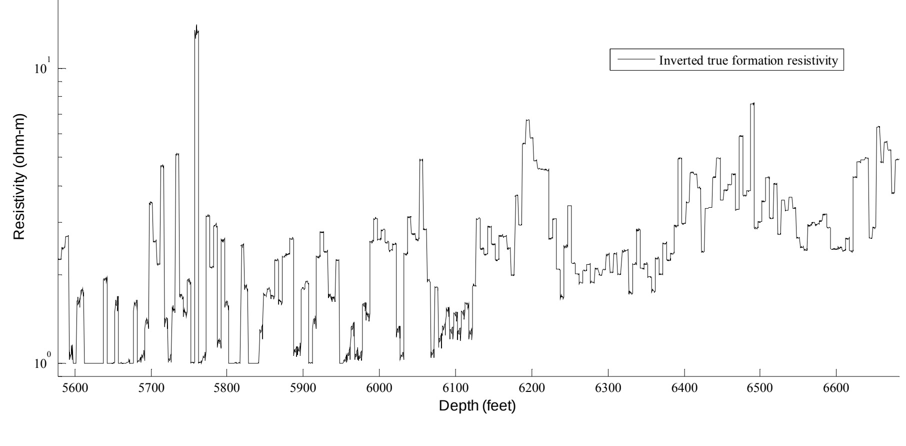


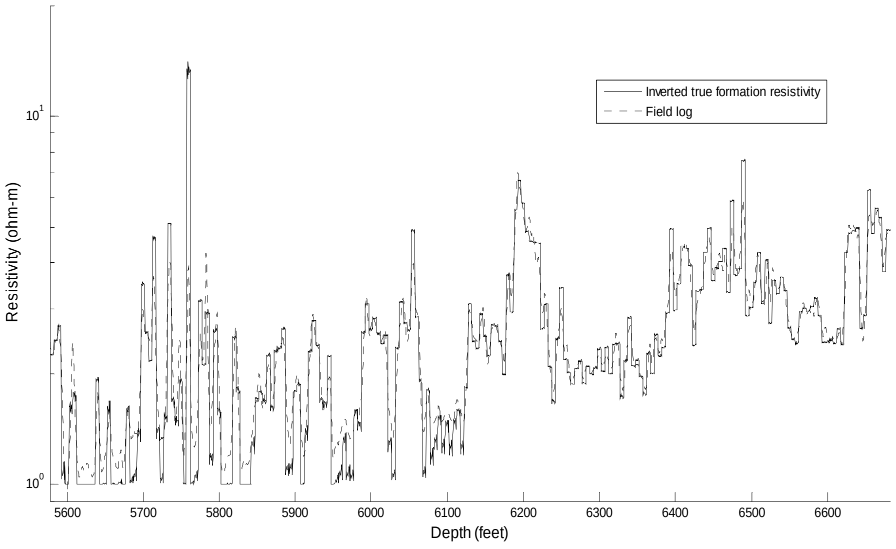
Fig. 6-5. The inverted true formation resistivity between 6,327.5 to 6,682 feet.



(a) The field log.



(b) The inverted true formation resistivity.



(c) Graph of field log and inverted true formation resistivity.

Fig. 6-6. The field logs and the inverted true formation resistivity.

7. Conclusions

In this study, we have adopted the MLP network and a hybrid method that combines MLP network and GA to apply on the well logging data inversion. MLP network is trained by gradient descent method and conjugate gradient method. We also expand the input features in MLP network to design 5 higher-order feature neural nets, and the testing result of higher-order feature neural nets is compared to the testing result of general MLP network.

In order to get more training pattern and better convergence result, each well logging dataset is split into sections. The number of features in each section is corresponding to the number of input nodes of MLP network. We have tested the MLP networks with 10, 20, 40, 100, and 200 input nodes respectively. Experimental results show that the network with 10 input nodes has the smallest average of mean absolute error. Besides, the performance of BP algorithm depends on the parameters and topology of neural networks, so we test the different learning rate, the number of hidden nodes, and the number of hidden layers. As our experiments, the one-hidden layer network with learning rate of 0.6, momentum parameter of 0.4, and the number of hidden nodes of 1.2 times the number of input nodes have the best performance. We then design 5 higher-order feature neural nets using the gradient descent method and conjugate gradient method. The experimental results show that the higher-order feature neural nets always have better performance than general MLP network that without higher-order features. The average training time is shorter and the mean absolute error is smaller. The network HOCG-3 that expands input features using second-order function and third-order function provides the largest improvement among all of 5 higher-order feature neural nets. The testing results presented show that our proposed higher-order feature neural nets using the conjugate gradient method is an effective alternative to well logging data inversion problem.

We also do the experiments on reversing the input and output of well logging data. This work is divided into two parts. First, we examine the inverse ability of MLP network using different number of input nodes with gradient descent method. The MLP network with 10 and 20 input nodes can not converged. However, the MLP network with 40 and 50 input nodes converged, and the MLP network with 50 input nodes has better performance, so we use 50 input nodes for HOML and

HOCG. Second, we do experiments on HOML and HOCG and compare the network performance between them. Experimental results show that the average performance of HOCG is better than HOML. Also we notice that the experiments on reversing the input and output of well logging data are more time-consuming than previous normal well logging data inversion tests.

In addition to the higher-order feature neural nets, we use the hybrid method that combines the neural network that using gradient descent and genetic algorithm. According to our testing result, it was shown that the performance of hybrid method has better performance than general MLP networks. Among the higher-order feature neural nets, HOML-3 with GA is the best combination that has the smallest average of mean absolute error. However, a disadvantage of using genetic algorithm in combination with neural network is the large computation cost required.

According to our experimental results, the network HOCG-3 has smallest average of mean absolute error, so we use HOCG-3 to do the well logging inversion which using the real field logs. The experimental result shows that our proposed higher-order feature neural net is an effective alternative to process the well logging data inversion.

ACKNOWLEDGEMENT

The author is grateful to Prof. Liang C. Shen for the well logging datasets, this paper will not be accomplished without these valuable datasets.

References

[1] P. Lippmann, "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine Volume 4, Issue 2, Part 1, pp. 4-22, April 1987.

[2] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*, PWS Publishing Co., Boston, 1996.

[3] L. S. Martin, D. Chen, and T. Hagiwara, "Neural Network Inversion of Array Induction Logging Data for Dipping Beds," SPWLA 42nd Annual Logging Symposium, Paper U1-U11, June 17, 2001.

[4] Z. H. Tan, "Hybrid Evolutionary Approach for Designing Neural Networks for Classification," Electronics Letters, Volume 40, Issue 15, pp. 955-957, 22 July 2004.

[5] W. Kinnebrock, "Accelerating the standard backpropagation method using a genetic approach," Neurocomput., Volume 6, pp. 583-588, 1994.

[6] C. P. Erick and K. Chandrika, "Evolving Neural Networks for the Classification of Galaxies," Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1019-1026, 2002.

[7] R. A. Dilruba, N. Chowdhury, F. F. Liza, and C. K. Karmakar, "Data Pattern Recognition Using Neural Network with Back-Propagation Training," International Conference on Electrical and Computer Engineering (ICECE), pp. 451-455, Dec. 2006.

[8] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network," International Joint Conference on Neural Networks (IJCNN), Volume 1, pp. 593-605, 18-22 June 1989.

[9] N. Qian, "On the momentum term in gradient descent learning algorithms," Neural Networks, Volume 12, Number 1, pp. 145-151, January 1999.

[10] M. Towsey, D. Alpsan, and L. Sztriha, "Training A Neural Network with Conjugate Gradient Methods," IEEE International Conference on Neural Networks, Volume 1, pp. 373-378, 27 Nov.-1 Dec. 1995.

[11] D. Goryn and M. Kaveh, "Conjugate Gradient Learning Algorithms for Multilayer Perceptrons," Proceedings of the Circuits and Systems 32nd Midwest Symposium, Volume 2, pp. 736-739, 14-16 Aug. 1989.

[12] J. R. Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain," School of Computer Science Carnegie Mellon University Pittsburgh, Aug. 4, 1994.

[13] C. Charalambous, "Conjugate Gradient Algorithm for Efficient Training of Artificial Neural Networks," IEE Proceedings on Circuits, Devices and

Systems, Volume 139, Issue 3, pp. 301-310, June 1992.

[14] C. H. Chen and H. Lai, "An Empirical Study of the Gradient Descent and the Conjugate Gradient Backpropagation Neural Networks," OCEANS '92. 'Mastering the Oceans Through Technology'. Proceedings Volume 1, pp. 132-135, October 26-29, 1992.

[15] R. Reed, "Pruning Algorithms-A Survey," IEEE Transactions on Neural Networks, Volume 4, Issue 5, pp. 740-747, Sept. 1993.

[16] K. Messer and J. Kittler, "Fast Unit Selection Algorithm for Neural Network Design," 15th International Conference on Pattern Recognition, Volume 2, pp. 981-984, 3-7 Sept. 2000.

[17] D. Whitley, "A Genetic Algorithm Tutorial," Statistics and Computing, Volume 4, Number 2, pp. 65-85, June 1994.

[18] J. Stender, "Introduction to Genetic Algorithms," Applications of Genetic Algorithms, IEE Colloquium, pp. 1/1-1/4, 15 Mar 1994.

[19] Y. Sasaki, H. Garis, and P. W. Box, "Genetic Neural Networks for Image Classification," IEEE International on Geoscience and Remote Sensing Symposium (IGARSS), Volume 6, pp. 3522-3524, 21-25 July 2003.

[20] D. X. Gone and X. G. Ruan, "A Neural Networks for Real Coded Genetic Algorithm," IEEE International Conference on Robotics, Intelligent Systems and signal Processing, Volume 2, pp. 874-879, 8-13 Oct. 2003.

[21] Z. Michalewicz, "Evolutionary Computation Techniques for Nonlinear Programming Problems," International Transactions in Operational Research, Volume 1, Number 2, pp. 223-240, 1994.

[22] X. Yao, "Evolving artificial Neural Networks," Proceedings of the IEEE, Volume 87, Number 9, pp. 1423-1447, Sep. 1999.