DOBD Algorithm for Training Neural Network: Part I. Method

WU Jian-yu(吴建昱)， HE Xiao-rong(何小荣)

(*Department of Chemical Engineering, Tsinghua University, Beijing 100084，China*)

Abstract: Overfitting is one of the important problems that restrain the application of neural network. The traditional OBD (Optimal Brain Damage) algorithm can avoid overfitting effectively. But it needs to train the network repeatedly with low calculational efficiency. In this paper, the Marquardt algorithm is incorporated into the OBD algorithm and a new method for pruning network－the Dynamic Optimal Brain Damage (DOBD) is introduced. This algorithm simplifies a network and obtains good generalization through dynamically deleting weight parameters with low sensitivity that is defined as the change of error function value with respect to the change of weights. Also a simplified method is presented through which sensitivities can be calculated during training with a little computation. A rule to determine the lower limit of sensitivity for deleting the unnecessary weights and other control methods during pruning and training are introduced. The training course is analyzed theoretically and the reason why DOBD algorithm can obtain a much faster training speed than the OBD algorithm and avoid overfitting effectively is given.

Key words: neural network; DOBD algorithm; Marquardt; overfitting; pruning; training; method
CLC No.：N945.12 **Document Code**：A **Article ID**：1009–606X(2002)02–0171–06

1 INTRODUCTION

Great progresses have been made on the research of Artificial Neural Network (ANN) in last decades. It has been proved theoretically that a Back–Propagation(BP) network has the capability to match any continuous function and there are some successful examples to apply ANN to chemical engineering[1]. However, there are still some problems such as overfitting that restrain the industrial application of ANN.

When an ANN is trained by samples, an important issue is how well it generalizes to samples outside the training sets. If the system does quite well during training but fails miserably when being checked with similar but slightly different inputs, we call this kind of phenomena as overfitting[2]. Generally speaking, several factors can easily cause overfitting. Firstly, samples of small number or with asymmetrical distribution may not exactly present enough mapping information. The mapping relation that is got through training with samples may easily deviate from the real one. Secondly, because of defects in measurement there is always noise in the samples especially in industrial data. Noise-contaminated samples carry less and inaccurate mapping information. Though the network can fit training samples with great precision, it may fail to other samples. Thirdly, a network with over-complicated framework usually has more free weights. This kind of network is likely to make the essentially simple mapping relation complicated because training samples are always discrete and finite in number.

It is another special issue to reduce the noise of samples. Using network with possibly few free weights can get better generalization without considering noise thus overfitting can be restrained

Received date: 2001-09-10, **Accepted date:** 2001-11-30
Biography: WU Jian-yu(1979–), male, native of Yangzhong city, Jiangsu province, MS, majoring in process system engineering.

effectively [3]. Since the network has fewer free weights, it has little possibility to complicate mapping relation.

Many methods to simplify network are presented in literatures and one kind used widely is pruning algorithms[2,4]. All these algorithms begin training with much more complex network, then determine and delete some weights or net units with less importance according to some rules. This research falls roughly into two categories. The first category involves adding an additional term that represents the complexity of network to the term of traditional error function of BP algorithm[2,5]. This additional term is E_{en} in Eq.(1):

$$C = \mu_{er} E_{er} + \mu_{en} E_{en}, \tag{1}$$

where E_{en} is the normal error function of BP algorithm. E_{er} and E_{en} converge gradually towards the minima respectively during the process of iteration. Although this kind of methods can overcome overfitting to a certain degree, it costs much more time for convergence because no superfluous weights are deleted and additional calculation of network complexity is carried out. Moreover, how to construct E_{en} and choose μ_{er} and μ_{er} are rather difficult.

The other aspect of research involves deleting some unimportant weights based on their sensitivities[6,7]. The OBD algorithm presented by Cun et al.[8] is a typical example. It presents a method to calculate sensitivity approximately with second-derivative information of error function and thus simplify network structure by deleting weights with low sensitivities. It shows good effect on avoiding overfitting. However, the network should complete a whole training process before the decision of deletion, which causes low efficiency of the algorithm.

We present a new algorithm called Dynamical Optimal Brain Damage (DOBD) on the basis of OBD algorithm. The Marquardt algorithm is combined into the new algorithm and a new method to simplify the network while training is also presented. Thus, the new algorithm has a high calculating efficiency and avoids overfitting efficiently.

2 OBD ALGORITHM

In the principle of the OBD algorithm[8], Cun et al assume that deleting some redundant weights from an over-complex network can improve generalization of network effectively and thus avoid overfitting. An important rule is that those weights to be deleted should have the least effects on training error, that is, the lowest sensitivities. So a reasonable strategy is to delete some small-sensitivity parameters while training the network. After deletion, the network should be trained again and this procedure can be iterated.

It would be prohibitively laborious to evaluate the sensitivity directly from its definition. Fortunately, it is possible to construct a local model of the error function and analytically predict the effect of perturbing the weight vector. It approximates the error function, E, by a Taylor series. A perturbation δU of the weight vector will change the error function by

$$\delta E = \sum_i g_i \delta u_i + \frac{1}{2}\sum_i h_{ii}\delta u_i^2 + \frac{1}{2}\sum_{i\neq j} h_{ij}\delta u_i \delta u_j + O\left\|\delta U\right\|^2, \tag{2}$$

where u_i is the components of weight vector U, g_i the components of the gradient of E with respect to U, and h_{ij} the elements of the Hessian matrix H of E with respect to U:

$$g_i = \frac{\partial E}{\partial u_i}, \qquad h_{ij} = \frac{\partial^2 E}{\partial u_i \partial u_j}.$$

However, it is also hard to calculate sensitivity directly according to Eq.(2) and some approximations are necessary. Firstly, it is assumed that δU caused by deleting several weights is the sum of the respective contributions by deleting a weight individually. So the third term of the right hand side of Eq.(2) is discarded. Secondly, parameter deleting will be performed after training has converged. The parameter vector is then at a (local) minimum of E and the first term of the right hand side of Eq.(2) can be neglected. Thirdly, the error function is nearly quadratic so that the last term in the equation can be neglected. Equation (2) then reduces to

$$\delta E = \frac{1}{2}\sum_i h_{ii}\delta u_i^2. \tag{3}$$

Every term of the right hand side of Eq.(3) presents the influence of a weight parameter on the error function. The OBD procedure is carried out as follows:

(1) Choose a reasonable network architecture;

(2) Train the network until a reasonable solution is obtained;

(3) Compute the second derivatives h_{kk} for each weight;

(4) Compute the sensitivity for each weight $s_k=h_{kk}u_k^2/2$;

(5) Sort the parameters by sensitivity and delete several weights with the lowest sensitivities;

(6) Iterate to step (2).

The OBD algorithm shows good effect on avoiding overfitting. However, some disadvantages limit its application: every time before pruning some weights, the network should complete a whole training process, which causes low efficiency of calculation.

3 DOBD ALGORITHM

The new algorithm, the Dynamic Optimal Brain Damage (DOBD), constitutes an improvement based on the OBD. The algorithm deletes low-sensitivity weights dynamically during the training process. As a result, the training efficiency can be improved greatly.

3.1 Adoption of the Maruqardt Algorithm

For the problem of minimization of $E(U)$:

$$\min \quad E(U)=F^{\mathrm{T}}(U)F(U), \tag{4}$$

with $U=(u_1, u_2, \cdots, u_n)^{\mathrm{T}}\in R^n$; $F(U)=[f_1(U), f_2(U), \cdots, f_m(U)]$.

The iterative equation of the Newton method is

$$U^{(k+1)} = U^{(k)} - H^{(k)^{-1}}\nabla E[U^{(k)}], \tag{5}$$

where $H^{(k)}$ is the Hessian matrix of $E(U)$ at point $U^{(k)}$. Since it is hard to calculate $H^{(k)}$ analytically, it is approximated by

$$H^{(k)} \approx 2J^{(k)^{\mathrm{T}}}J^{(k)}, \tag{6}$$

where $J^{(k)}$ is the Jacobian matrix, $J_{i,j}^{(k)} = \dfrac{\partial f_i[U^{(k)}]}{\partial u_j}$. Since $\nabla E[U^{(k)}]$ can be shown as

$$\nabla E[U^{(k)}] = 2U^{(k)T}F[U^{(k)}], \tag{7}$$

the iterative equation of the Marquardt method can be described as[9]

$$U^{(k+1)} = U^{(k)} + \Delta U^{(k)}, \qquad \{[J^{(k)\mathrm{T}} J^{(k)}] + \lambda I\}\Delta U^{(k)} = -J^{(k)\mathrm{T}} F[U^{(k)}], \tag{8}$$

where I is an identity matrix, λ is a regulative parameter to be determined dynamically during the process of training.

It has been established that the Marquardt algorithm can offer a much faster speed of convergence on the aspect of training BP–ANN than the traditional BP algorithm[10], which is an important reason for incorporating the Marquardt algorithm into the DOBD. Additionally, its adoption can simplify the calculation of sensitivity more effectively than other method, as described in the following.

3.2 Calculate Sensitivity Dynamically

The kernel of the new algorithm is the method through which sensitivities can be calculated dynamically. As described in the OBD algorithm, the variation of the training error is

$$\delta E \approx \sum_i g_i \delta u_i + \frac{1}{2}\sum_i h_{ii}\delta u_i^2 . \tag{9}$$

There are two reasons for which the first term of the right hand side of Eq.(9) can not be ignored as in OBD. Firstly, the training process may not have converged into the local minimal point when deleting weights begins. In fact some weights must be deleted before the training process is over. Secondly, the oscillation caused by deleting weights while training maybe also makes a significant influence on g_i.

According to Eq.(9), the expressions to forecast the change of error function caused by deleting a certain weight can be concluded:

$$\Delta E_i = g_i u_i + \frac{1}{2} h_{ii} u_i^2 , \tag{10}$$

where ΔE_i is just the sensitivities of weights u_i. According to Eqs.(6) and (7), g_i is just the components of vector ∇E and h_{ij} is the elements of Hessian matrix. Consequently, all the parameters that need to calculate sensitivity have been already calculated in the Marquardt iteration process. Additional calculation is saved, which improves the speed of this new algorithm.

The DOBD procedure can be carried out as follows:

(1) Take some iteration of training by the Marquardt method optimization;

(2) Compute the sensitivity for each weight $\Delta E_i = g_i u_i + 1/2\, h_{ii} u_i^2$;

(3) Delete some weights with sensitivities lower than limit;

(4) Take some more iterations of training;

(5) Iterate to step (2).

4 CONTROL FACTORS OF DOBD

Although the procedure of DOBD is given above, there are some important control factors that may make remarkable influence on the result of DOBD.

4.1 Lower Sensitivity Limit for Deleting Weights

A crucial problem of dynamically pruning is that to what degree we should simplify the network. It is hard to prognosticate the finial effect of pruning as the OBD method that can checkout the effect by another full training. Lower sensitivity limit (LSL) for deleting weights is used as an alternative way to control the pruning process. Every time after some iterations, the sensitivities of all weights are calculated respectively and some of the weights whose sensitivities are below LSL are deleted. After

enough iterations and prunings, sensitivities of remained weights are almost all above LSL. So the pruning process tends to end and the final construction of the network is thought to be more optimal.

The particularity of Marquardt algorithm, which is pointed out by Sjoberg et al.[11] in research on separable non-linear least squares minimization, makes it possible to get a current value of LSL. Since $J^{\mathrm{T}}J+\lambda I$ [in Eq.(8)] is symmetric it can be separated as

$$J^{\mathrm{T}}J+\lambda I = T^{\mathrm{T}}\,\mathrm{diag}(q_1+\lambda,\Lambda\ \Lambda\ ,q_n+\lambda)T\ , \tag{11}$$

where $\{q_k\}$ is the eigenvalues of $J^{\mathrm{T}}J$. Since $J^{\mathrm{T}}J$ is positive semi-definite, $q_k\geq 0$. $\Delta U^{(\mathrm{k})}$can be solved from Eq.(8)

$$\Delta U^{(k)} = -J^{\mathrm{T}}F[U^{(k)}](J^{\mathrm{T}}J+\lambda I)^{-1} = -J^{\mathrm{T}}F[U^{(k)}]T^{\mathrm{T}}\,\mathrm{diag}\left(\frac{1}{q_1+\lambda},\Lambda\ \Lambda\ ,\frac{1}{q_n+\lambda}\right)T. \tag{12}$$

An important feature of Eq.(12) is: Given $\lambda>0$, for those convergence directions corresponding to small eigenvalues $\lambda>>q_k$, the Gauss–Newton method gives a large step size $1/q_k$, and the Marquardt method on the other hand gives a small step of size $1/(q_k+\lambda)\approx 1/\lambda$. For those convergence directions corresponding to large eigenvalues $\lambda<<q_k$, the Marquardt method gives a step size $1/(q_k+\lambda)\approx 1/\lambda$ approximately equal to the Gauss–Newton method. In this way Marquardt divides the parameters of weights into two sub-classes. Within the first class one has an efficient convergence of nearly Gauss–Newton and thus a high sensitivity, and within the second class one has a slow convergence of nearly the steepest descent method and a low sensitivity.

Figure 1 is an example that shows the particular distribution of weights sensitivities. 200 samples are generated from function $y=\sin x$, where x is the pseudo-random number and $x\in[-2\pi, 2\pi]$. A three-layer ANN of 50 hidden units is used for training with the Marquardt algorithm. In the low sensitivity (less than about 0.05) region, the distribution is more concentrated than others. These weights can be considered as low sensitivity ones. In the high sensitivity (above about 0.1) region, the distribution is comparatively uniform. It is also found that this kind of distribution is relatively stable during the first 20~30 iterations and also has no marked variation with respect to different examples. So it is possible to set LSL at 0.01~0.1.

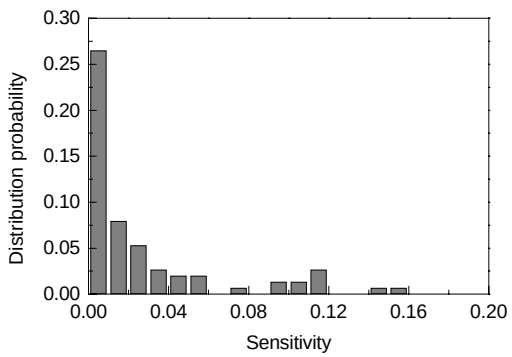

Fig.1 Distribution probability of weights based on different sensitivities after 20 iterations

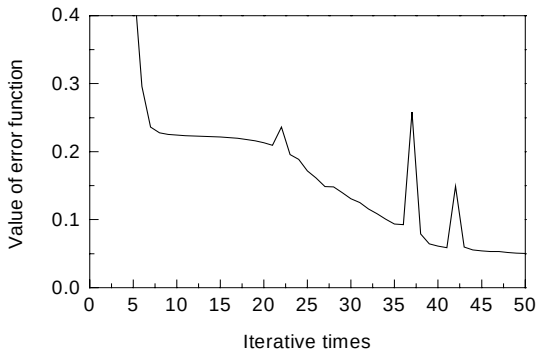

Fig.2 Oscillation on error function during pruning training

4.2 Interval of Pruning

There will be an overshooting of error function every time after pruning. Consequently there should

be an optimal number of iterations before the next pruning. Moreover, the original sensitivity distribution of weights has been destroyed and the new one has not been stabilized just after pruning and another immediate pruning may cause improper deletion of weights. However, the Marquardt method can eliminate this oscillation fast. Training ANN with the same samples and the same network of Fig.1, from the 15^{th} iteration, 10 weights with sensitivities less than 0.1 have been deleted every 5 iterations. The change on error function is described in Fig.2. Although after pruning there may be a remarkable change on training error, after one iteration the error can be approximately decreased to the previous level. So about 3 iterations is enough between two consecutive pruning processes.

4.3 Maximum One-time Pruning Number of Weights

The maximum one-time pruning number (MOPN) is defined as the maximum allowed number of deleted weighs at a time during the pruning process, although from Fig.2 MOPN is not a keen factor for DOBD for the great capability of eliminating overshooting. However, if MOPN is too small, it will take a long time to delete all the redundant weights and on the other hand deleting too many weights once may make the subsidence of oscillation fail and consequently lead to a much high value of final error. We suggest that the value of MOPN is about 10~50, depending on the total number of weights in the initial network structure.

5 CONCLUSIONS

A new algorithm called DOBD that combines the Marquardt algorithm with the traditional OBD has been presented and the principle of the new algorithm described. It uses the particularity of the Marquardt method so that both the training and the pruning processes can go along synchronously without repetition of the whole training process. Several important factors are discussed to make the new algorithm more efficient and more practical.

REFERENCES:

[1] YAO X L. Research on the Application of ANN on the Optimal Operation of Petroleum Chemical Engineeringv [D]. Beijing: Tsinghua Univ., 1993, 65–89 (in Chinese).

[2] Reed R. Pruning Algorithms-A Survey [J]. IEEE Transactions on Neural Networks, 1993, 4(5): 740–747.

[3] Kindermann J, Linden A. Practical Complexity Control in Multilayer Perceptrons [J]. Signal Processing, 1999, 74(1): 29–46.

[4] Ponnapalli P V S, Ho K C, Thomson M. Formal Selection and Pruning Algorithm for Feedforward Artificial Neural Network Optimization [J]. IEEE Transactions on Neural Networks, 1999, 10(4): 964–968.

[5] Chauvin Y. A Back-propagation Algorithm with Optimal Use of Hidden Units [A]. Touretzky D S. Advances in Neural Information Processing(2) [C]. Denver: Morgan Kaufmann, 1990. 519–526.

[6] Fletcher A P, Cloete L. Variance Analysis of Sensitivity Information for Pruning Multilayer Feedforward Neural Networks [A]. Engelbrecht. Proceedings of the International Joint Conference on Neural Networks [C]. New York: IEEE, 1999. 1829–1833.

[7] Kijsirikul B, Chongkasemwongse K. Decision Tree Pruning Using Backpropagation Neural Networks [A]. Proceedings of the International Joint Conference on Neural Networks [C]. Washington DC: Phathumwan, 2001. 1876–1880.

[8] Cun Y Le, Denker J S. Optimal Brain Damage [A]. Touretzky D S. Advances in Neural Information Processing(2) [C]. Denver: Morgan Kaufmann, 1990. 598–605.

[9] CHEN B L. Algorithms and Theories of Optimization [M]. Beijing: Tsinghua University Press, 1989. 383–389 (in Chinese).

[10] Hagan M T, Menhaj M B. Training Feedforward Networks with the Marquardt Algorithm [J]. IEEE Transactions on Neural Networks, 1994, 5(6): 989–993.

[11] Sjoberg J, Viberg M. Separable Non-linear Least-squares Minimization-possible Improvements for Neural Net Fitting [A]. Proceedings of the 1997 IEEE Signal Processing Society Workshop [C]. New York: IEEE, 1997. 345–354.