International Symposium on Nonlinear Theory and its Applications
Xi'an, PRC, October 7-11, 2002

Optimization of the Backpropagation Algorithm for Training Feedforward Neural Networks

Tsang-Min Lin[1], Teng-Chiao Lin[2], Ker-Wei Yu[3] and Chien-Cheng Yu[4]

[1]Department of Mechanical Engineering
Hsiuping Institute of Technology
[2] Institute of Technological and Vocational Education
National Taipei University of Technology
[3] Department of Marine Engineering
National Kaohsiung Institute of Marine Technology
[4]Department of Electrical Engineering
Hsiuping Institute of Technology

Abstract: The standard backpropagation algorithm is widely recognized as a powerful tool for training multilayer feedforward neural networks (MFNNs). But since it applies the steepest descent method to update the weights, it suffers from a slow convergence rate and often yields suboptimal solutions. In this paper, we attempt to propose a simple method to take advantage of both the steepest descent and Newton methods to speed up convergence. From the simulation results, it is shown that the proposed method can provide stable learning and improved convergence rate.

Key words: Backpropagation algorithm; Convergence; Steepest descent; Newton's method

1. Introduction

The backpropagation (BP) algorithm [1] is one of the most popular training algorithms for multilayer feedforward neural networks (MFNNs). Unfortunately, it can be very slow for practical applications. The MFNNs supervised learning process is equivalent to a nonlinear unconstrained optimization problem, where a global error function is minimized via the parameters adjustment of the neural network [2]-[4]. Several of methods adapted from numerical analysis have been applied in an attempt to use both the gradient of the error function and the second derivative in constructing efficient supervised training algorithms to accelerate the convergence of the algorithm [5]-[8].

The main numerical analyses procedures that can be implemented computationally use methods that require only the evaluation of the local gradient of the function, or methods that use also the second order derivatives. In the first case, the function is approximated by its constant and linear terms of its Taylor's expansion, and in the second case, the quadratic term is also considered [9]-[12]. The steepest descent method reduces the function value when the design vector w^k is away from the optimum point w^*. On the other hand, the Newton method converges fast when the design vector w^k is close to the optimum point w^*. Under ideal circumstances the convergence rate of Newton's method is quadratic, whereas steepest descent has linear convergence. However, at points that are remote from the optimum point, modifications to the Hessian may be necessary to ensure positive definiteness of this matrix and to ensure descent [5]. Furthermore, training algorithms that apply nonlinear conjugate gradient methods, such as the the Polak-Ribiere (PR) methods [9], or quasi-Newton methods, such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [12] are computationally intensive for MFNNs with several hundred weights. In addition, it is not certain that the extra computational cost speeds up the minimization process for nonconvex functions when far from a minimizer, as is usually the case with the neural network training problem [10].

In this paper, we attempt to propose a simple method to take advantage of both the steepest descent and Newton methods to speed up convergence. The remaining of the paper is organized as follows: In Section 2 we investigate some of the existing numerical optimization techniques. In Section 3, we propose a simple method to take advantage of both the steepest descent and Newton methods to speed up convergence. Experiments and simulation results are presented in Section 4. Finally, we make a conclusion.

2. The Standard Numerical Optimization Techniques

Basically, Backpropagation is a gradient descent

technique to minimize some error criteria E. Most descent methods cannot locate the minimum of a function in a finite number of steps. The method of steepest descent can be defined as:

$$w^{k+1} = w^k - \alpha_k g^k \tag{1}$$

where $g^k \equiv \nabla E(w^k)$, and α_k is a nonnegative scalar that minimizes $E(w^k - \alpha g^k)$. Steepest descent searches from the point w^k along the direction of the negative gradient $-g^k$ to a minimum point on this line, where the minimum point is taken as w^{k+1}. The method of steepest descent may appear to be the best unconstrained minimization technique since each one-dimensional search starts in the best direction. However, due to the fact that the steepest descent direction is a local property, the method is not really effective in most applications [7]. The method of steepest descent usually works quite well during early stages of the optimization process. However, as a stationary point is approached, the method usually behaves poorly, where small orthogonal steps are taken [9].

The idea behind Newton's method is that the energy function being minimized is approximated locally by a quadratic function, and this approximate function is minimized exactly. In order to motivated the procedure, consider the following quadratic approximation q at a given point w^k:

$$q(w) = E(w^k) + \nabla E(w^k)^T (w - w^k) + \frac{1}{2}(w - w^k)^T H(w^k)(w - w^k) \tag{2}$$

where $H(w^k)$ is the Hessian matrix of E at w^k. A necessary condition for a minimum of the quadratic approximation q is that

$$\nabla q(w) = 0 \tag{3}$$

or

$$\nabla E(w^k) + H(w^k)(w - w^k) = 0 \tag{4}$$

Assuming that the inverse of $H(w^k)$ exists, the successor point w^{k+1} is given by

$$w^{k+1} = w^k - H(w^k)^{-1} \nabla E(w^k) \tag{5}$$

or

$$w^{k+1} = w^k - H(w^k)^{-1} g^k \tag{6}$$

The above equation gives the recursive form of the points generated by Newton's method for the multidimensional case. In general, the points generated by the method of Newton may not converge. The reason is that $H(w^k)$ may be singular, so that w^{k+1} is not well defined. Even if $H(w^k)^{-1}$ exists, $E(w^{k+1})$ is not necessarily less than $E(w^k)$. However, if the starting point is close enough to a point w^* such that $\nabla E(w^*) = 0$ and $H(w^*)$ is of full rank, then the method of Newton is well defined and converge to w^* [10].

The quasi-Newton methods iteratively use successively improved approximations to the inverse Hessian instead of the true inverse as in the basic Newton method. The sequential quasi-Newton methods employ the differences of two successive iteration points and the difference of the corresponding gradients to approximate the inverse Hessian matrix. In quasi-Newton methods the matrices $H(w^k)$ are positive definite approximations of the inverse Hessian matrix obtained solely from gradient information. Therefore, it is not required to evaluate second-order derivatives, the algorithms are always stable since g^k is always a descent search direction. However, they require the evaluation and storage in memory of a totally dense matrix $H(w^k)$ at each iteration step, so that for a large number of variables the storage requirements are extremely large. One of the most powerful quasi-Newton methods is the BFGS algorithm which can be formulated as

$$w^{k+1} = w^k + \alpha_k g^k \tag{7}$$

$$g^k \cong w^{k+1} - w^k = -H(w^k)\nabla E(w^k) \tag{8}$$

The conjugate gradient algorithm is something of a compromise; it dose not require the calculation of second derivatives, and yet it still has the quadratic convergence property [8]. It converges to the minimum of a quadratic function in a finite number of iterations. The conjugate gradient algorithm in its simplest form can be formulated as

$$w^{k+1} = w^k + \alpha_k g^k \tag{9}$$

$$g^k = \beta_k g^{k-1} - \nabla E(w^k) \tag{10}$$

where the scalars β_k can be chosen by several different methods, which produce equivalent results for quadratic functions [9]-[12]. The conjugate gradient methods require much less storage than the quasi-Newton methods.

However, they require an exact determination of the learning rate α_k and the parameter β_k in each iteration step. Moreover, the conjugate gradient methods require approximately twice as many gradient evaluations as the quasi-Newton methods, but they save time and memory needed for computing $H(w^k)$ for large problems.

3. The Proposed Learning Algorithm

It is clear that training feedforward neural networks to minimize squared error is simply a numerical optimization problem. The Newton's method is a well-established numerical optimization technique with quadratic speed of convergence. However, the practical application of the Newton's method to MFNNs is not recommended due to the fact that the exact calculation of the Hessian matrix and its inversion are very computational intensive [11].

The steepest descent method reduces the function value when the design vector is away from the optimum point. The Newton's method, on the other hand, converges fast when the design vector is close to the optimum point. In this section, we attempt to take advantage of both the steepest descent method and the Newton's method. This method combines both the methods as

$$w^{k+1} = w^k - [\lambda_k \alpha_k I + (1-\lambda_k) H^{-1}(w^k)] g^k \qquad (11)$$

where $\lambda_k \in (0,1)$. The value of λ_k is taken to be large at the beginning and then reduced to zero gradually as the iterative process progresses. Thus, as the value of λ_k decreases from a large value to zero, the characteristics of the search method change from those of a steepest descent method to those of the Newton's method.

Assuming that $a_k = \lambda_k \alpha_k$, $b_k = 1-\lambda_k$, $b_k \neq 0$, and substituting into Eq. (11) leads to

$$w^{k+1} = w^k - [a_k I + b_k H^{-1}(w^k)] g^k \qquad (12)$$

Set $\sigma_k = a_k / b_k$, then update the weights of the network according to

$$w^{k+1} = w^k - b_k [\sigma_k I + H^{-1}(w^k)] g^k \qquad (13)$$

4. Simulation results

In this section, we are going to present one example of application to illustrate the problem of approximating the function $\sin(x)\cos(2x)$. The function $\sin(x)\cos(2x)$ was trained with 21 learning samples over the period $[0,2\pi]$. The simulations have been carried out on IBM PC using MATLAB version 5 and the performance of the proposed algorithm has been evaluated and compared with the PR method, BFGS method, and DFP method. For all the algorithms, the desired sum squared error is SSE=0.1, the number of hidden units are 10, the maximum number of training epochs is 500, the mean value of the final uncertainty interval for the golden section method is equal to 0.1%, and the weights were initialized uniformly over the interval [-0.5, 0.5].

Fig.1 and Fig.2 present the error behavior and the resultant approximation, given by the BP algorithm, respectively. Fig.3 shows the learning curves of these algorithms.

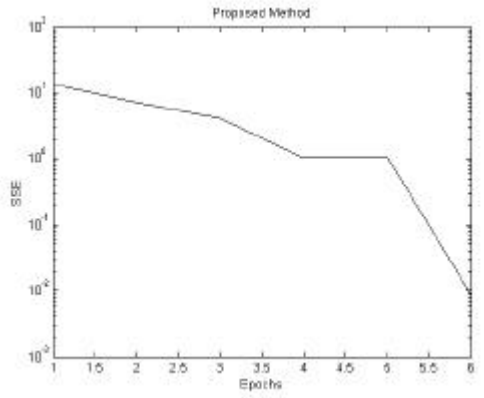

Fig. 1 Error behavior

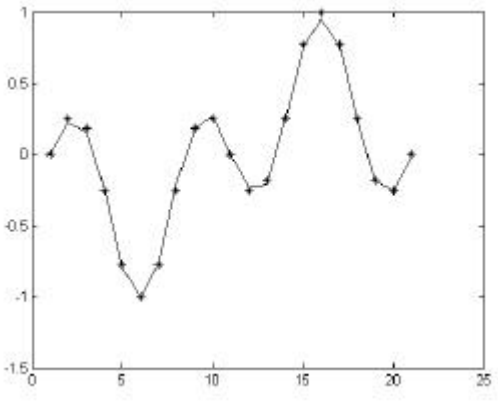

Fig. 2 Resultant approximation

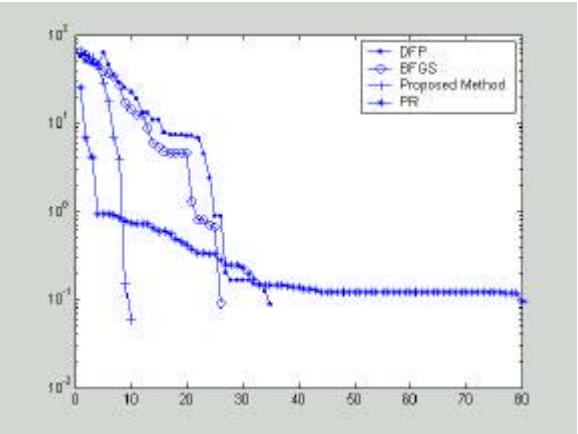

Fig. 3 The learning curves

5. Conclusions

The steepest descent method has the great advantage of being very simple and cheap to implement. However, it may exhibit extremely slow convergence. On the other hand, the Newton's method converges fast when the design vector is close to the optimum point, but generally they are rather expensive to implement. Moreover, Newton's methods involve the evaluation of the inverse Hessian matrix which often introduces computational difficulties and singularity problems. In this paper, we propose a simple method to take advantage of both the steepest descent and Newton methods to speed up convergence. As compared to some powerful methods, the proposed algorithm has the advantages of fast convergence speed and low computational complexity.

Reference

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

[2] R. J. Schalkoff, *Artificial Neural Networks*, New York: McGraw-Hill, 1997.

[3] S. Russell, and P. Norvig, *Artificial Intelligence-A Modern Approach*, Englewood Cliffs, NJ: Prentice-Hall, 1995.

[4] X. -H. Yu, G. -A. Chen, and S. -X. Cheng, "Dynamic learning rate optimization of the backpropagation algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 669-677, May 1995.

[5] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[6] A. Cichocki, and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, New York: John Wiley & Sons, 1993.

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

[8] F. M. Ham, and I. Kostanic, *Principles of Neurocomputing for Science & Engineering*, New York: McGraw-Hill, 2000.

[9] D. G. Luenberger, *Linear and Nonlinear Programming*, 2nd ed., Addison-Wesley, 1989.

[10] M. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming – Theory and Algorithms*, 2nd ed., John Wiley & Sons Inc., 1993.

[11] E. Polak, *Optimization: Algorithms and Consistent Approximations*, Springer-Verlag, 1997.

[12] S. S. Rao, *Engineering Optimization: Theory and Practice* , 3rd ed., John Wiley & Sons Inc., 1996.