Step Acceleration Based Training Algorithm for Feedforward Neural Networks

Yanlai Li [a], Kuanquan Wang [a], and David Zhang [b]
[a] *Biometrics Research Center, Dept. of Computer Science and Engineering*
Harbin Institute of Technology (HIT), Harbin, 150001, P.R. China
[b] *Department of Computing,Hong Kong Polytechnic University, Hong Kong*

Abstract

This paper presents a very fast step acceleration based training algorithm (SATA) for multilayer feedforward neural network training. The most outstanding virtue of this algorithm is that it does not need to calculate the gradient of the target function. In each iteration step, the computation only concentrates on the corresponding varied part. The proposed algorithm has attributes in simplicity, flexibility and feasibility, as well as high speed of convergence. Compared with the other methods, including the conventional BP, the conjugate gradient (CG), and the BP based on weight extrapolation (BPWE), many simulations have confirmed the superiority of this algorithm in terms of converging speed and computation time required.

1. Introduction

The most influential effort in all learning methods of artificial neural network (ANN) is the development of back-propagation (BP) algorithm, which is a gradient descent with a fixed learning rate [1] and has a drawback with slow convergence and time consuming.

In order to accelerate the BP algorithm, a number of improved algorithms have been described in the literature. Several heuristic rules are proposed for adapting the learning rates [2, 3]. Moreover, modified error functions, which are different from popular mean-squared errors, show a faster convergence through decreasing the possibility of a premature saturation [4, 5]. In spite of this improved convergence, these methods are still based on the gradient descent approach. This will inevitably increase the computing time or lead to failure of convergence [9].

For improving ANN's performance, second-order nonlinear optimizing methods are then used. Such several learning algorithms have been developed, for example, the convenient formulations for the computation of the second-order derivatives of the error function are introduced in both the case of sigmoid functions [6] and a more general case [7]. Ideally, this kind of method should possess the fastest convergence. However, it may suffer from three problems: 1) ill conditioning of the Hessian matrix; 2) time-consuming processing in calculating the first-order derivation of the state vector to the performance index; and 3) estimating the inverse Hessian matrix [8, 9].

A universal acceleration technique for the BP algorithm based on weight extrapolation (BPWE) is proposed in [10]. This extrapolation procedure is easy to be implemented since it is activated only a few times between iterations of the conventional BP algorithm. This leads to significant savings in computation time of the standard BP algorithm. However, BPWE lies much on the characteristic of BP. When BP behaves surging, it is difficult to process an extrapolation. In this paper, we present a fast algorithm, SATA, which greatly accelerates the convergence for feed-forward ANN. It is evolved from the step acceleration algorithm in optimizing theory [14]. The propounded algorithm reserves the good attributes, such as the robust-ness of BP. Moreover, it uses the corresponding coordinate rotation to avoid calculating the derivative of the error function; hence its equivalent error equation is much simplified. In this way, this algorithm has the advantage of both fast convergence and simple computation.

This paper is organized as follows: The new training algorithm is defined in Section 2. Theory analysis of computational complexity of BP and SATA is given. Computer simulations and performance comparisons are discussed in Section 3. At last, section 4 gives a brief conclusion.

2. SATA Descriptions

For the sake of simplicity, let us assume that the neural network that we are going to discuss contains only one hidden layer, and the number of nodes of the input layer, output layer and hidden layer is I, O, H, respectively. In addition, the number of the training patterns is defined as K. The mean-square-error function (MSE) of the network is:

1051-4651/02 $17.00 © 2002 IEEE

$$MSE = \frac{1}{2}\frac{1}{K}\sum_{p=1}^{K}\sum_{i=1}^{O}\left(y_{pi} - d_{pi}\right)^2, \quad (1)$$

where y_{pi} is the actual output of the ith output node with regard to the pth training pattern, while d_{pi} is the corresponding desired output.

Step acceleration is an easy-programming and effective method for optimizing problems with fewer variables. Generally, it is composed of two alternate steps: searching around and moving ahead. The aim of the search part is to find a more optimal point around the basis point, i.e., to find a point to make the error decrease. The coordinate rotation approach is used here to fulfill searching, which is one of the oldest multidimensional searching methods. It processes the searching along every coordinate in turn while other coordinates remain fixed.

With respect to the three-layered structure, there are four classes of parameters should be adjusted: the thresholds of the output layer units; the weights from the output layer units to the hidden layer ones; the thresholds of the hidden layer units; and the weights from the hidden layer units to the input layer ones. Here, the sigmoid function is adopted as the activation function, i.e.,

$$g(x) = \frac{1}{1+e^{-\lambda x}}, \quad (2)$$

where λ is the parameter which decides the shape of sigmoid function.

Let $E = \sum_p \sum_i \left(y_{pi} - d_{pi}\right)^2$. We take the thresholds of output units as example and give the deduction of SATA.

2.1. Searching Around

A. Thresholds of output units

Fixing other parameters, the error function of output unit's threshold modification can be defined as

$$\begin{aligned} E_A &= \sum_p \sum_i \left(g\left(\sum_j w_{ij} b_{pj} + \theta_i \right) - d_{pi} \right)^2 \\ &= \sum_p \sum_{i \neq i_0} \left(g\left(\sum_j w_{ij} b_{pj} + \theta_i \right) - d_{pi} \right)^2 \\ &\quad + \sum_p \left(g\left(A_1 + \theta_{i_0}\right) - d_{pi_0} \right)^2, \end{aligned} \quad (3)$$

where θ_i is the threshold of the ith output unit, d_{pi_0} is the desired output of the i_0 th output unit corresponding with the pth training pattern, w_{ij} is the weight between the ith output unit and the jth hidden one, b_{pj} is the output of the jth hidden unit, and $A_1 = \sum_j w_{i_0 j} b_{pj}$ is the total input of the i_0 th output unit. Change of θ_{i_0} only causes the change of the second part in Eq. (3) which has relation with θ_{i_0}. Since other variables, including A_1 and y_{pi}, can be acquired by transferring the storage of the last iteration, we can use an equivalent error function to compute the error function, i.e.,

$$E_A' = \sum_p \left(g\left(A_1 + \theta_{i_0}\right) - d_{pi_0} \right)^2. \quad (4)$$

Other parameters can be deducted in the same way as the first case. The equivalent error functions are listed as follows respectively:

B. Weights from Output Units to Hidden Units

$$\begin{aligned} E_B &= \sum_p \sum_{i \neq i_0} \left(g\left(\sum_j w_{ij} b_{pj} + \theta_i \right) - d_{pi} \right)^2 \\ &\quad + \sum_p \left(g\left(A_2 + B_2 w_{i_0 j_0}\right) - d_{pi_0} \right)^2, \end{aligned} \quad (5)$$

where $A_2 = \sum_{j \neq j_0} w_{i_0 j} b_{pj} + \theta_{i_0}$, and $B_2 = b_{pj_0}$.

$$E_B' = \sum_p \left(g\left(A_2 + B_2 w_{i_0 j_0}\right) - d_{pi_0} \right)^2. \quad (6)$$

C. Thresholds of Hidden Units

$$\begin{aligned} E_C &= \sum_p \sum_i \left(g\left(\sum_j w_{ij} g\left(\sum_k \overline{w}_{jk} \overline{b}_{pk} + \overline{\theta}_j \right) + \theta_i \right) - d_{pi} \right)^2 \\ &= \sum_p \sum_i \left(g\left(A_3 + B_3 g\left(C_3 + \overline{\theta}_{j_0}\right)\right) - d_{pi} \right)^2, \end{aligned} \quad (7)$$

where $A_3 = \sum_{j \neq j_0} w_{ij} b_{pj} + \theta_i$, $B_3 = w_{ij_0}$, $C_3 = \sum_k \overline{w}_{j_0 k} \overline{b}_{pk}$; $\overline{\theta}_j$ is the threshold of the jth hidden unit, $\overline{w}_{kj}$ is the weight between the kth input unit and the jth hidden unit, and $\overline{b}_{pk}$ is the input of the k th input unit.

D. Weights from Hidden Units to Input Units

$$\begin{aligned} E_D &= \sum_p \sum_i \left(g\left(\sum_j w_{ij} g\left(\sum_k \overline{w}_{jk} \overline{b}_{pk} + \overline{\theta}_j \right) + \theta_i \right) - d_{pi} \right)^2 \\ &= \sum_p \sum_i \left(g\left(A_4 + B_4 g\left(C_4 + D_4 \overline{w}_{j_0 k_0}\right)\right) - d_{pi} \right)^2, \end{aligned} \quad (8)$$

where $A_4 = \sum_{j \neq j_0} w_{ij} b_{pj} + \theta_i$, $B_4 = w_{ij_0}$,

$C_4 = \sum_{k \neq k_0} \overline{w}_{j_0 k} \overline{b}_{pk} + \overline{\theta}_{j_0}$, $D_4 = \overline{b}_{pk_0}$.

It is clear that our problem has been much simplified after the above steps because it can be solved by any linear search approach.

2.2. Moving Ahead

Now, a vector, $V = [w_{ij}, \overline{w}_{jk}, \theta_i, \overline{\theta}_j]^T$, can be obtained. It is a natural thought that the target function would be declined if we go along the direction of $V - V'$ (where V' is the point of last iteration). That is just the idea of step acceleration that we are going to apply to further optimization. The whole algorithm is summarized as:

Begin

Initialize the network parameters, $\theta_{i0}, \overline{\theta}_{j0}, w_{ij0}, \overline{w}_{jk0}$, with random value in (-1, 1)

Define computational accuracy, ε

$V_0 \leftarrow [w_{ij0}, \overline{w}_{jk0}, \theta_{i0}, \overline{\theta}_{j0}]^T$

Initialize the iteration counter, $IC = 0$

$Iter\_Err \leftarrow Error(V_0)$

Do

{

$V_1 \leftarrow search(V_0)$ // using (4, 6, 7, 8)

$Iter\_Err1 \leftarrow Error(V_1)$

$Iter\_Err \leftarrow Iter\_Err1$

If $Iter\_Err < \varepsilon$ Then Exit End If

$V_0 \leftarrow V_1 + \alpha * (V_1 - V_0)$. //Generally, $\alpha = 1$

$V_2 \leftarrow search(V_0)$

$Iter\_Err2 \leftarrow Error(V_2)$

If $Iter\_Err2 < Iter\_Err1$ Then

$V_0 \leftarrow V_2 + \beta * (V_2 - V_1)$ //Generally, $\beta = 1$

$Iter\_Err \leftarrow Error(V_0)$

$IC++$

Else

$V_0 \leftarrow V_1$

$Iter\_Err \leftarrow Iter\_Err1$

$IC++$

End If

} While ($Iter\_Err > \varepsilon$)

End

2.3. Analysis of Computational complexity

The algorithm described in this section has the most remarkable virtue of fast convergence. In fact, the computational complexity of BP has been proved in [13], which is exponential function of pattern numbers. Our computational complexity can be given as follows.

Theorem - *The computational complexity of SATA is exponential function of the scale of network parameters.*

For example, assume that the number of nodes of the input layer, output layer and hidden layer for the network is *I*, *O*, *H*, respectively. The number of the training patterns is *K*. It needs run $(I * H + O * H + O) * K$ times multiplication and the sigmoid function $(H + O) * K$ times when an error function is called in BP algorithm. While it only needs run multiplication $(3 + 8 * O) * K$ times and call sigmoid function $(2 + 2 * O) * K$ times.

It is evident that much saving of computations in SATA than in BP according to the theory above.

3. Experiments and comparisons

In this section, examples including XOR problem, approximation of function $f(x) = \frac{2}{5}\sin(2\pi x) + \frac{1}{2}$, and the iris classification problem are taken to demonstrate the performance of SATA. For a comparison, simulation results of the conventional BP, the conjugate gradient (CG) algorithm [12], and the BPWE algorithm are also carried out. All these experiments are programmed in language C and run on PC III 600. Results are shown in the Fig. 1- Fig. 4. In the BP learning, the learning rate η is fixed to 0.8 to gain the best result.

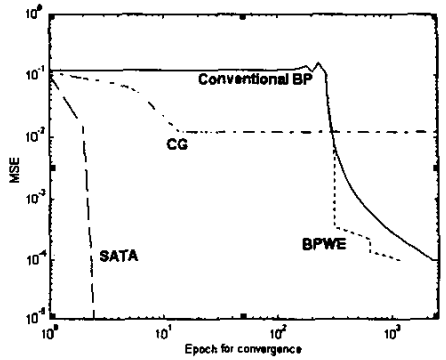

Figure 1. Contrast learning curve for XOR

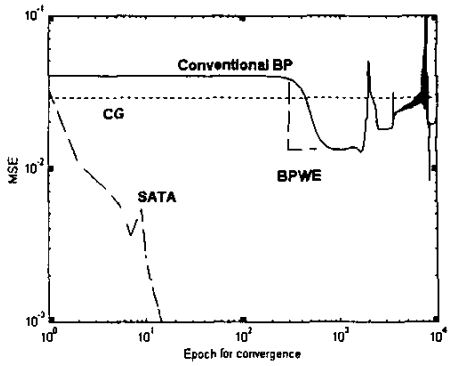

Figure 2. Contrast learning curve for approximation problem

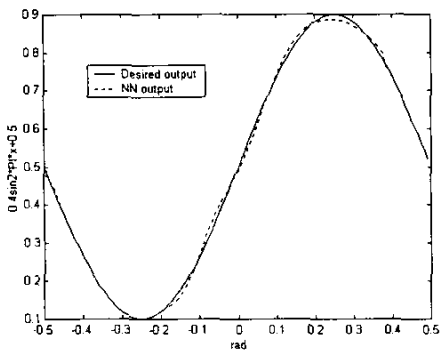

Figure 3. Approximating result of SATA

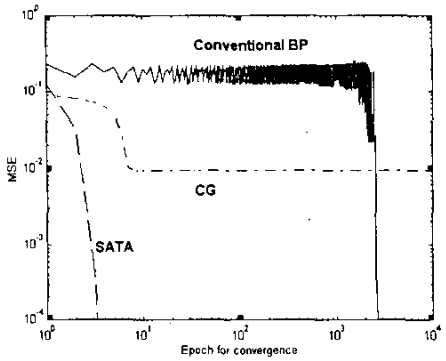

Figure 4. Contrast learning curve for a three-class classification problem

4. Conclusions

A learning algorithm is judged on the basis of certain, rather conflicting requirements, such as simplicity, flexibility and efficiency [11]. In this paper, we have developed a new and fast algorithm, SATA, for multiplayer feedforward ANN. It yields results in both accuracy and convergence rates, which are orders of magnitude superior compared to the conventional BP, CG and BPWE algorithm. Besides, It is a linear optimizing problem and need not calculate the derivative of error function. Furthermore, there are no learning parameters to be tuned by the user once the network structure is constructed. The fact that the computation complexity is exponential function of the network parameters scale suggests that this algorithm is suitable for the problems with simple network structure and large number of patterns, which are more popular in practical applications.

Acknowledgements

The research was partly supported by National Natural Science Foundation (NSFC)/High-Tech 863-306 fund and The Central/Departmental Fund of Hong Kong Polytechnic University.

References

[1] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing*. Cambridge, MA: MIT Press (1986).

[2] R.A. Jacobs, "Increased rates of convergence through learning rate adaptation", *Neural networks*, 1, 295-307 (1988).

[3] T.P. Vogl, J.K. mangis, A.K. Rigler, W.T. Zink, and D.L. Alkon, "Accelerating the convergence of the back-propagation method¡", *Biol. Cybern*, 59, 257-263 (1988).

[4] S.H. Oh, "Improving the error backpropagation algorithm with a modified error function", *IEEE Trans. on Neural Networks*, 8, 799-803 (1997).

[5] A.V. Oyen and B. Nienhuis, "Improving the convergence of the back-propagation algorithm", *Neural Networks*, 5, 465-471 (1992).

[6] C. Bishop, " Extract calculation of Hessian matrix for the multilayer perceptron", *Neural Computa.*, 4, 494-501 (1992).

[7] W.L. Buntine and A.S. Weigend, "Computing second derivatives in feed forward networks: A review", *IEEE Trans. on Neural Networks*, 5, (1994).

[8] R. Parisi, E.D. Di Claudio, G. Orlandi, and B.D. Rao. "A generalized learning paradigm exploiting the structure of feed forward neural networks", *IEEE Trans. on Neural Networks*, 7, 465-471 (1996).

[9] G.J. Wang and C.C. Chen, "A fast multilayer neural network training algorithm based on the layer-by-layer optimizing procedures", *IEEE Trans. on Neural Networks*, 7, 768-775 (1996).

[10] S.V. Kamarthi and S. Pittner, "Accelerating neural network training using weight extrapolations", *Neural Networks*, 12, 1285-1299 (1999).

[11] N.B. Karayiannis and A.N. Venetsanopoulos, "Fast learning algorithms for neural networks", *IEEE Tran. Circuits and Systems- II: Analog and Signal Processing*, 39, 453-474 (1992).

[12] M. Pfister and R. Rojas, "Speeding-up backpropagation- a comparison of orthogonal techniques", *Proceedings of the International Joint Conference on Neural Networks*, 1. Nasgoya, Japan: Japanese Neural Network Society, 517-523 (1993).

[13] L. Zhang and B. Zhang, "On the BP algorithm of feedforward neural networks", *International Journal of Pattern Recognition and Artificial Intelligence*, 7, 191-195 (1994).

[14] Xue Jiaqing, Optimization: Theory and Method (revised edition), *Metallurgy Industry Publishing House*, 1992.