Numerical Integration Based on a Neural Network Algorithm

The authors present a neural network approach to numerical integration and propose four methods for training neural networks. Their results show that their own numerical integration method has high accuracy and a fast convergence rate, making its application values significant for engineering practice.

Integration is important in science and technology. Scholars all over the world have presented various numerical integration approaches[1–5] that have effectively solved lots of practical engineering problems. The digital proportion-integration-differentiation (PID) controller, for instance, is concerned with numerical integration. To calculate the definite integration's value, researchers often apply the Newton-Leibniz formula

$$\int_a^b f(x)dx = F(b) - F(a), \quad (F'(x) = f(x)), \qquad (1)$$

but in many cases, the function $f(x)$ is difficult or complicated to get. Besides, in engineering practice, the function $f(x)$ is presented as a function table rather than an analytic expression, which means we can't always use the Newton-Leibniz formula.

Alternatives to this formula include the Newton-Cotes, Romberg, and Gauss methods.[6–8] Among them, the Newton-Cotes method is the most common approach to establishing numerical integration with an interpolation polynomial. However, because the convergence of the high-degree Newton-Cotes method is poor, researchers seldom use it in practical calculations. As for the Romberg method, its convergence speed is fast, and its precision is high, but it requires intensive calculation. Finally, the Gauss method's accuracy is high, it's numerically stable, and its convergence speed is fast, but calculating the node and coefficient is complicated. It also requires knowledge of the analytic expression of function $f(x)$.

In this article, we present a numerical-integration method based on neural networks. Our main idea is to make neural networks' outputs fit the function $f(x)$ by training weights with cosine basis functions. Because quadrature of such functions is easy, we can approximately consider the numerical integration of any function $f(x)$ as the integration of cosine basis functions. It's well known that the back-propagation (BP) algorithm isn't fit for adjusting large-scale weights, and although it minimizes unstable behavior, it usually fails. To effectively solve these problems, we introduce the *momentum*,[9,10] *conjugate gradient* (CG),[9,11,12] and *truncated Newton* (TN)[13–15] methods for training neural networks.

Algorithm Description

Figure 1 illustrates a neural network model based

1521-9615/06/$20.00 © 2006 IEEE
Copublished by the IEEE CS and the AIP

Zeng Zhe-Zhao and Wang Yao-Nan
Hunan University, China
Wen Hui
Changsha University of Science & Technology, China

on cosine basis functions. We can see that the network consists of one input layer, one hidden layer, and one output layer. The input layer has only one input vector $\mathbf{X}$ with $(N + 1)$ elements; the hidden layer consists of an intelligent agent that has $(N + 1)$ neurons; and the output layer has only one output vector $\mathbf{Y}$ with $(N + 1)$ elements. The activation function of the hidden layer unit is a vector $\mathbf{C}^T\mathbf{W}$ with $(N + 1)$ elements, where $\mathbf{W}$ is a weight vector of the neural network with $(N + 1)$ elements, $\mathbf{C}$ is an activation matrix of the hidden units, and $\mathbf{F}$ is the neural network's desired output vector.

The following algorithm shows that all of the neural network's sample data are synchronously trained in the form of a vector and not trained one by one. Therefore, the neural network we introduce here is obviously different from other neural networks.

If $\mathbf{W} = [w_0, w_1, \ldots, w_N]^T$, $\mathbf{X} = [x_0, x_1, \ldots, x_N]^T$, $\mathbf{Y} = [y(x_0), y(x_1), \ldots, y(x_N)]^T$, $\mathbf{F} = [f(x_0), f(x_1), \ldots, f(x_N)]^T$,

and

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \cos(x_0) & \cos(x_1) & \ldots & \cos(x_N) \\ \vdots & \vdots & \ldots & \vdots \\ \cos(Nx_0) & \cos(Nx_1) & \ldots & \cos(Nx_N) \end{bmatrix}, \quad (2)$$

then we can approximately express the function $f(x)$ in Equation 1 as

$$y(x) = \sum_{n=0}^{N} w_n \cos(nx) \quad (3)$$

or

$$\mathbf{Y} = \mathbf{C}^T\mathbf{W}, \quad (4)$$

where $x \in [0, \pi]$. We define an objective function $\mathcal{J}$ as

$$\mathcal{J} = \frac{1}{2}\left\|\mathbf{E}^k\right\|^2, \quad (5)$$

where

$$\mathbf{E}^k = \mathbf{F} - \mathbf{Y}^k. \quad (6)$$

Here, $\mathbf{E}^k$ is the error vector between the neural network's desired and actual outputs, and $\mathbf{F}$ is the neural network's desired output. To minimize $\mathcal{J}$, we compute $\mathbf{W}^k$ recursively via a simple gradient descent rule,

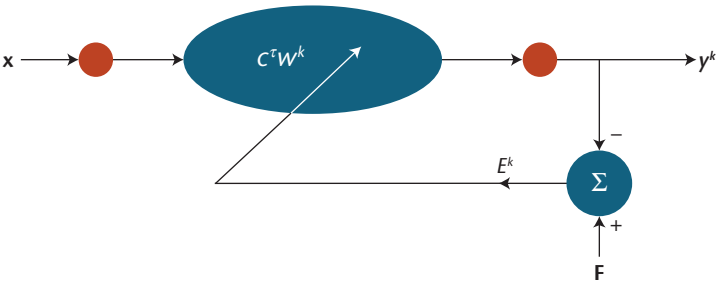

Figure 1. The neural network model of the fitting function $f(x)$. The network consists of an input layer, a hidden layer, and an output layer.

$$\mathbf{W}^{k+1} = \mathbf{W}^k - \eta\frac{\partial\mathcal{J}}{\partial\mathbf{W}^k}, \quad (7)$$

where $\eta > 0$ is a learning rate. Differentiating Equation 5 with respect to $\mathbf{W}^k$, we get

$$\frac{\partial\mathcal{J}}{\partial\mathbf{W}^k} = \frac{\partial\mathbf{Y}^k}{\partial\mathbf{W}^k}\frac{\partial\mathbf{E}^k}{\partial\mathbf{Y}^k}\frac{\partial\mathcal{J}}{\partial\mathbf{E}^k} = -\mathbf{C}\mathbf{E}^k. \quad (8)$$

Substituting Equation 8 into Equation 7, we have

$$\mathbf{W}^{k+1} = \mathbf{W}^k + \eta\mathbf{C}\mathbf{E}^k. \quad (9)$$

To ensure the neural network's absolute convergence, it's important to select a proper learning rate η.

Here's a proof for the neural network's convergence.

Theorem 1: When $0 < \eta < 2/\|\mathbf{C}\|^2$, the neural network algorithm is convergent, where η is the learning rate and $\mathbf{C}$ is a matrix produced by cosine basis functions. See Equation 2.

Proof: Define a Lyapunov function,

$$V_k = \frac{1}{2}\left\|\mathbf{E}^k\right\|^2. \quad (10)$$

Therefore, we have

$$\Delta V_k = \frac{1}{2}\left\|\mathbf{E}^{k+1}\right\|^2 - \frac{1}{2}\left\|\mathbf{E}^k\right\|^2 \quad (11)$$

and

$$\mathbf{E}^{k+1} = \mathbf{E}^k + \Delta\mathbf{E}^k = \mathbf{E}^k + \frac{d\mathbf{E}^k}{d\mathbf{W}^k}\Delta\mathbf{W}^k \quad (12)$$

because

$$\Delta\mathbf{W}^k = -\eta\frac{\partial\mathcal{J}}{\partial\mathbf{W}^k} = \eta\mathbf{C}\mathbf{E}^k, \quad \frac{d\mathbf{E}^k}{d\mathbf{W}^k} = -\mathbf{C}.$$

According to Equations 11 and 12, we have

$$\begin{aligned}\Delta V_k &= \frac{1}{2}\left\|(\mathbf{I}-\eta\mathbf{C}^2)\mathbf{E}^k\right\|^2 - \frac{1}{2}\left\|\mathbf{E}^k\right\|^2 \\ &\le \frac{1}{2}\left\|\mathbf{I}-\eta\mathbf{C}^2\right\|^2\left\|\mathbf{E}^k\right\|^2 - \frac{1}{2}\left\|\mathbf{E}^k\right\|^2 \\ &= \frac{1}{2}\left[\left\|\mathbf{I}-\eta\mathbf{C}^2\right\|^2 - 1\right]\left\|\mathbf{E}^k\right\|^2. \end{aligned} \quad (13)$$

Because

$$\left\|\mathbf{E}^k\right\|_2^2 > 0,$$

if the algorithm is convergent—that is, $\Delta V_k < 0$—it's easy to see from Equation 13 that

$$\left\|\mathbf{I}-\eta\mathbf{C}^2\right\|_2^2 - 1 < 0$$

or

$$\left\|\mathbf{I}-\eta\mathbf{C}^2\right\|^2 < 1, \quad (14)$$

where $\mathbf{I}$ is a unit matrix and $\|\bullet\|^2$ is the square of the Euclidean norm.

Because $\|\mathbf{I}-\eta\mathbf{C}^2\|^2 \ge (1-\eta\|\mathbf{C}\|^2)^2$, considering $\eta > 0$ and $\|\mathbf{C}\|^2 > 0$ according to Equation 14, we have $(1-\eta\|\mathbf{C}\|^2)^2 < 1$ or $0 < \eta < 2/\|\mathbf{C}\|^2$.

Apparently, if the learning rate η satisfies $0 < \eta < 2/\|\mathbf{C}\|^2$, then we see from Equation 12 that $\Delta V_k < 0$. Therefore, the algorithm is convergent and we prove the theorem completely.

In our experience, when the learning rate η is equal to

$$(0.65 \sim 0.70)\times\frac{2}{\|\mathbf{C}\|^2} = \frac{1.3\sim 1.4}{\|\mathbf{C}\|^2},$$

it's optimal. Hence, in this article, we consider the optimal learning rate to be $\eta_{opt} = 1.35/\|\mathbf{C}\|^2$.

We can now use Equation 9 in a computer program to adjust the neural network's weight vector $\mathbf{W}$ and summarize the whole algorithm as follows:

1. Sample uniformly the function $f(x)$ to obtain the neural network's training sample vector $\mathbf{F}$—that is, $\{\mathbf{X}, \mathbf{F}\}$, where $\mathbf{X} = [x_0, x_1, \ldots, x_N]^T$ and $\mathbf{F} = [f(x_0), f(x_1), \ldots, f(x_N)]^T$—produce the random weight vector $\mathbf{W}$, and define an arbitrarily small positive real number *Tol*. Produce the activation matrix $\mathbf{C}^T$ of the neural network's hidden units, and let $\eta = 1.35/\|\mathbf{C}\|^2$.
2. Produce the new predicted neural network output vector $\mathbf{Y}^k$ using Equation 4.
3. Compute the error vector $\mathbf{E}^k$ and $\mathcal{J}$ using Equations 5 and 6.
4. Update the weight vector $\mathbf{W}^{k+1}$ using the BP algorithm according to Equation 9.
5. If $\mathcal{J} > Tol$, go to Step 2. Otherwise, stop the neural network's training.

It's well known that the BP algorithm has a local-minimization problem. To improve the neural network's convergence, we can introduce three optimal approaches to train the neural network's weight vector.

Neurocomputing and Optimization

As we mentioned earlier, the BP algorithm isn't fit for adjusting large-scale weights. Furthermore, although it minimizes unstable behavior, the BP algorithm isn't convergent. To effectively solve these problems, we introduce three optimization methods.

Momentum Training Algorithm

We know from the last section that the standard BP algorithm implements the steepest descent method. At each step, the weight vector is adjusted in the direction in which the error function decreases most rapidly. This direction is determined by the gradient of the error surface at the current point in the weight vector space. Using the error's gradient, we can update the weight vector adjustments for the connections in a negative direction to the gradient with a certain rate, which Equation 9 gives.

To prompt the neural network's learning, we use the momentum training algorithm.[9] Here, we modify the error function so that a portion of the previous weight vector is fed through to the current weight vector. Hence, the weight vector is updated according to

$$\mathbf{W}^{k+1} = \mathbf{W}^k + \eta\mathbf{C}\mathbf{E}^k + \lambda\Delta\mathbf{W}^k, \quad (15)$$

where λ is the momentum factor ($0 < \lambda < 1$).

Conjugate Gradient Training Algorithm

Because learning in realistic neural network applications often involves adjusting several hundred weights, only optimization approaches applicable to large-scale problems are relevant as alternative learning algorithms. The numerical analysis community's general opinion is that CG methods are well suited to effectively handle large-scale problems.[9,11,12] The CG algorithm combines the advantages of the steepest descent method's simplicity and better convergence with-

out evaluating, inverting, and storing the Hessian matrix **H**. Among unconstrained optimization approaches, the CG method is perhaps the easiest one that applies to large-scale problems.[12]

To minimize $\mathcal{J}$, we update the weight vector $\mathbf{W}^k$ according to a CG rule:

$$\mathbf{W}^{k+1} = \mathbf{W}^k + \alpha(k)\mathbf{P}(k), \tag{16}$$

where $\mathbf{P}(k)$ is a descent direction and $\alpha(k)$ is the adaptive learning rate with respect to iteration k.

We can describe the CG algorithm as follows. Per Equation 8, define $\mathbf{g}(k) = \nabla\mathcal{J}(k) = -\mathbf{CE}^k$:

$$\alpha(k) = \frac{\mathcal{J}(k)}{\|\mathbf{g}(k)\|^2},$$

and

$$\beta(k) = \frac{\|\mathbf{g}(k)\|^2}{\|\mathbf{g}(k-1)\|^2}.$$

Here, $\beta(k)$ is a conjugate search rate. Set $\mathbf{P}(0) = -\mathbf{g}(0) = \mathbf{CE}^0$, set $k = 0$, and then compute $\mathbf{P}(k) = -\mathbf{g}(k) + \beta(k)\mathbf{P}(k-1)$. The weight vector is updated according to $\mathbf{W}^{k+1} = \mathbf{W}^k + \alpha(k)\mathbf{P}(k)$.

Truncated Newton Training Algorithm

We know from previous research[9,13–15] that the TN method has a fast convergence rate and is especially fit for large-scale optimization problems. If numerical integration is implemented via a neural network with the TN training algorithm, the network's convergence rate will be faster than that of a neural network with the steepest descent and CG methods.

To minimize $\mathcal{J}$, we update the weight vector $\mathbf{W}^k$ according to the TN rule

$$\mathbf{W}^{k+1} = \mathbf{W}^k + \alpha(k)\mathbf{d}(k), \tag{17}$$

where $\mathbf{d}(k)$ is a search direction and $\alpha(k)$ is a search step rate.

We can describe the TN algorithm as follows:

1. Define $\mathbf{g}(k) = \nabla\mathcal{J}(k) = -\mathbf{CE}^k$ (see Equation 8) and $\mathbf{H}(k) = \nabla^2\mathcal{J}(k)$, where $\mathbf{g}(k)$ is a gradient direction and $\mathbf{H}(k)$ is the Hessian of the cost function $\mathcal{J}(k)$.
2. Set $\mathbf{p}(0) = 0$, $\mathbf{r}(0) = -\mathbf{g}(k)$, $\mathbf{d}(0) = \mathbf{r}(0)$, $\delta_0 = \|\mathbf{r}(0)\|^2$, and $j = 0$.
3. Let $\mathbf{q}(j) = \mathbf{H}(k)\mathbf{d}(j)$. If $\mathbf{d}^T(j)\mathbf{q}(j) \le \varepsilon\delta_j$, stop and output
$$\mathbf{d}(k) = \begin{cases} \mathbf{d}(0), & j = 0 \\ \mathbf{p}(j), & j > 0. \end{cases}$$
4. Let
$$\alpha(j) = \frac{\|\mathbf{r}(j)\|^2}{\mathbf{d}^T(j)\mathbf{q}(j)},$$
$\mathbf{p}(j+1) = \mathbf{p}(j) + \alpha(j)\mathbf{d}(j)$, and $\mathbf{r}(j+1) = \mathbf{r}(j) - \alpha(j)\mathbf{q}(j)$. If $\|\mathbf{r}(j+1)\| \le \mu(k)\|\mathbf{g}(k)\|$, stop and output $\mathbf{d}(k) = \mathbf{p}(j+1)$, where $\mu(k) = \min\{1/k, \|\mathbf{g}(k)\|\}$.
5. Let
$$\beta(j) = \frac{\|\mathbf{r}(j+1)\|^2}{\|\mathbf{r}(j)\|^2},$$
$\mathbf{d}(j+1) = \mathbf{r}(j+1) + \beta(j)\mathbf{d}(j)$, $\delta(j+1) = \|\mathbf{r}(j+1)\|^2 + \beta^2(j)\delta(j)$, and $j = j+1$.

Go to Step 3.

Optimization approaches applicable to large-scale problems are relevant as alternative learning algorithms.

Neural Network Weights

To compute the numerical integration of function $f(x)$, we propose the numerical integration theorem using neural network weights.

Theorem 2: Let $f(x) \in C[a, b]$, $0 \le a, b \le \pi$, and $\mathbf{W} = [w_0, w_1, \ldots, w_N]^T$ be the neural networks' weight vector. If

$$y(x) = \sum_{n=0}^{N} w_n \cos(nx)$$

is the neural networks' output, then

$$I = \int_a^b f(x)dx \approx (b-a)w_0 + \sum_{n=1}^{N} \frac{1}{n} w_n[\sin(nb) - \sin(na)]. \tag{18}$$

Proof:

$$I = \int_a^b f(x)dx \approx \int_a^b y(x)dx = \int_a^b \sum_{n=0}^{N} w_n \cos(nx)dx$$

Table 1. The results of example 1.

$f(x)$	x^2	x^4	$1/(x+1)$	$\sqrt{1+x^2}$	$\sin(x)$	e^x
Exact value	2.667	6.400	1.099	2.958	1.416	6.389
Trapezoidal	4.000	16.000	1.333	3.326	0.909	8.389
Simpson's	2.667	6.667	1.111	2.964	1.425	6.421
Neural network rule	2.667	6.400	1.099	2.958	1.415	6.389
Training iterations	9	12	10	10	10	13
Neural network momentum training ($\lambda = 0.06$)	2.667	6.400	1.099	2.958	1.415	6.388
Training iterations	8	9	7	7	7	8
Neural network conjugate gradient rule	2.667	6.400	1.099	2.958	1.415	6.388
Training iterations	6	7	6	6	6	6
Neural network truncated Newton rule	2.667	6.400	1.099	2.958	1.416	6.389
Training iterations	4	4	4	4	4	4

$$= \int_a^b [w_0 + \sum_{n=1}^{N} w_n \cos(nx)]dx$$

$$= (b-a)w_0 + \sum_{n=1}^{N} w_n \int_a^b \cos(nx)dx$$

$$= (b-a)w_0 + \sum_{n=1}^{N} \frac{1}{n} w_n [\sin(nb) - \sin(na)].$$

According to this theorem, we can make three inferences:

- If $a = 0$ and $b < \pi$, then

$$I = \int_a^b f(x)dx = \int_0^b f(x)dx$$

$$= bw_0 + \sum_{n=1}^{N} \frac{1}{n} w_n \sin(nb). \quad (19)$$

- If $a = 0$ and $b = \pi$, then

$$I = \int_a^b f(x)dx = \int_0^\pi f(x)dx = \pi w_0. \quad (20)$$

- If $a > 0$ and $b = \pi$, then

$$I = \int_a^b f(x)dx = \int_a^\pi f(x)dx$$

$$= (\pi - a)w_0 - \sum_{n=1}^{N} \frac{1}{n} w_n \sin(na). \quad (21)$$

Results and Discussion

Two examples from related research[8] can help us compare the neural network (NN), neural network momentum training (NNM), neural network conjugate gradient training (NNCG), and truncated Newton training (NNTN) algorithms.

Example 1

The trapezoidal rule for a function $f(x)$ on the interval [0, 2] is

$$\int_0^2 f(x)dx \approx f(0) + f(2),$$

and Simpson's rule for $f(x)$ on the interval [0, 2] is

$$\int_0^2 f(x)dx \approx \frac{1}{3}[f(0) + 4f(1) + f(2)].$$

In this article, our rule for $f(x)$ on the interval [0, 2] is

$$\int_0^2 f(x)dx \approx 2w_0 + \sum_{n=1}^{N} \frac{1}{n} w_n \sin(2n),$$

where if the length of weight vector $\mathbf{W}$ is equal to 30, then $N = 29$. If we produce a matrix $\mathbf{C}$ using Equation 2 and the random weight vector $\mathbf{W}$ with 30 elements, we can define an arbitrary small positive real number $Tol = 10^{-6}$. We can then make the activation matrix $\mathbf{C}^T\mathbf{W}$ of the neural network's hidden units and let $\eta_{opt} = 1.35/\|\mathbf{C}\|^2$.

Table 1 summarizes the results to seven places for some elementary functions. We see distinctly from Table 1 that all the results we obtained using our four rules (NN, NNM, NNCG, and NNTN) are better than reported elsewhere.[8] Ultimately, the results we obtained with the NNTN rule are the

Table 2. The results of example 2.

$f(x)$	$I = \int_0^{48} \sqrt{1+(\cos x)^2}\,dx$	J	Training iterations
Composite Simpson's	58.47028		
Neural network rule	58.47046	3.883×10^{-17}	24
Neural network momentum training ($\lambda = 0.06$)	58.47046	7.495×10^{-17}	15
Neural network conjugate gradient rule	58.47046	4.371×10^{-17}	12
Neural network truncated Newton rule	58.47046	3.386×10^{-17}	6

best. The results in Table 1 show that the NNTN method has a faster convergence rate and is especially fit for the large-scale optimization problems.

Example 2

Compute the following approximation to

$$\int_0^{48} \sqrt{1+(\cos x)^2}\,dx.$$

From previous work,[8] we know that this integral causes difficulty with the Bomberg integration, which uses the composite trapezoidal rule to give preliminary approximations and then applies the Richardson extrapolation process to improve the approximations. When we use a composite Simpson's rule—subdividing the interval $[a, b]$ into n subintervals and applying Simpson's rule on each consecutive pair of subintervals—we divide the interval [0, 48] into 100 subintervals, and the result is 58.47028.[8]

Considering the function has a period equal to π, and $48 = 15\pi + 0.8761$, according to Theorem 2, our rule for $f(x)$ on the interval [0,48] is

$$\int_0^{48} \sqrt{1+(\cos x)^2}\,dx \approx 15\pi w_0 + 0.8761 w_0 + \sum_{n=1}^{N} \frac{1}{n} w_n \sin(0.8761n),$$

where if the length of weight vector $\mathbf{W}$ is equal to 30, then $N = 29$. We can define an arbitrary small positive real number $Tol = 10^{-16}$ and learning rate $\eta_{opt} = 1.35/\|\mathbf{C}\|^2$.

Table 2 summarizes the results. To verify their validity, we use the adaptive Lobatto quadrature function QUADL in Matlab to compute the integral of the function

$$f(x) = \sqrt{1+\cos^2 x}$$

in the interval [0,48]. The result is

$$\int_0^{48} \sqrt{1+(\cos x)^2}\,dx \approx 58.47047\ .$$

All the results we obtained using our four rules (NN, NNM, NNCG, and NNTN) are better than those reported elsewhere.[8] We see again that the NNTN method has a faster convergence rate.

Our work here establishes a new application of neural networks that, combined with advanced optimization algorithms to numerical integration, offers a great opportunity for additional development and improvements such as designing digital PID control.

Of course, the neural network model we present in this article has some limitations. For example, the variable x in Equation 3 is limited from 0 to π—that is, $x \in [0, \pi]$—so the quadrature interval of function $f(x)$ is also limited from 0 to π. To extend the quadrature interval of the $f(x)$ and satisfy the needs of science and engineering practice, our neural network model will need further improvement. CiSE

References

1. X. Hua and Y. Guoxiao, "Numerical Integration Method for a Class of Oscillating Functions," *J. Beijing Univ. of Science & Technology*, vol. 19, no. 3, 1999, pp. 280–284.
2. G. Hanwei et al., "The Numerical Integration Algorithm Based on Multiresolution Analysis," *J. Nat'l Univ. of Defense Technology*, vol. 22, no. 4, 2000, pp. 94–97.
3. L. Cheng-Zhi and C. Dou-Xing, "An Efficient Step-Size Control Method in Numerical Integration for Astrodynamical Equations," *ACTA Astronomica Sinica*, vol. 43, no. 4, 2002, pp. 387–390.
4. L. Weidong et al., "Precise Numerical Integration of Nonlinear Rotor System with Multi-Degree of Freedom," *J. Vibration Eng.*, vol. 17, no. 4, 2004, pp. 427–432.
5. Z. Yan-Hong, C. Shan-Yang, and H. Xiao, "A Numerical Integration Method for Structural Analysis," *Eng. Mechanics*, vol. 22, no. 3, 2005, pp. 39–45.

6. S. Jian-Hua, *Foundation of Numerical Method*, TongJi Univ. Press, 1999, pp. 73–109.
7. W. Neng-Chao, *Numerical Analysis*, Higher Education Press, 1997, pp. 66–96.
8. R.L. Burden and J.D. Faires, *Numerical Analysis*, 7th ed., Thomson Learning, 2001, pp. 186–226, 772.
9. M.S. Al-Haik, H.Garmestani, and I.M. Navon, "Truncated-Newton Training Algorithm for Neurocomputational Viscoplastic Model," *Computer Methods in Applied Mechanics and Eng.*, vol. 192, no. 19, 2003, pp. 2249–2267
10. X. Jun, L. Zhenzhong, and W. Jun, "Diagnostic Model of Cinder Characteristic Based on Improved BP Algorithm," *J. Eng. for Thermal Energy and Power*, vol. 17, no. 3, 2002, pp. 271–274.
11. C. Chunhong, Z. Yongjian, and L. Wenhui, "A New Method for Geometric Constraint Solving: A Hybrid Genetic Algorithm Based on Conjugate Gradient Algorithm," *Chinese J. Scientific Instrument*, vol. 25, no. 4, 2004, pp. 389–392.
12. W. Huaxiang, Z. Xueming, and Z. Lifeng, "Conjugate Gradient Algorithm for Electrical Capacitance Tomography," *J. Tianjin Univ.*, vol. 38, no. 1, 2005, pp. 1–4.
13. L. Ximing and C. Zixing. "Active Set Truncated-Newton Algorithm for Large-Scale Bound Constrained Minimization," *J. Central South Univ. of Technology*, vol. 33, no. 1, 2002, pp. 82–86.
14. L. Ximing and Q. Jixin, "Subspace Truncated-Newton Algorithm for Large-Scale Bound Constrained Optimization," *J. Zhejiang Univ.*, vol. 29, no. 5, 2002, pp. 494–499.
15 R.S. Dembo and T. Steihaug, "Truncated-Newton Algorithm for Large-Scale Unconstrained Optimization," *Mathematical Programming*, vol. 26, no. 2, 1983, pp. 190–212.

Zeng Zhe-Zhao *is a professor in the College of Electric & Information Engineering at Changsha University of Science & Technology, China, and is a doctoral graduate student at Hunan University, China. His research interests include signal processing, neural networks, computer and control, and communication. Zhe-Zhao has an MS in physics from Tsinghua University, China. Contact him at hncs6699@yahoo.com.cn.*

Wang Yao-Nan *is a professor in the College of Electric & Information Engineering at Hunan University, China. His research interests include artificial intelligence and signal processing. Yao-Nan has a PhD in automation from the National Defence University of Science & Technology, China. Contact him at yaonan@mail.hunu.edu.cn.*

Wen Hui *is a vice professor in the College of Electric & Information Engineering at Changsha University of Science & Technology, China. His research interests include signal processing and neural networks. Hui has an MS in circuits and systems from the National Defence University of Science & Technology, China. Contact her at 00710154@163.com.*