Speeding up the scaled conjugate gradient algorithm and its application in neuro-fuzzy classifier training

Article *in* Soft Computing · October 2009

DOI: 10.1007/s00500-009-0410-8 · Source: DBLP

2 authors:

Bayram Cetisli
T.C. Süleyman Demirel Üniversitesi
37 PUBLICATIONS **229** CITATIONS
SEE PROFILE

Atalay Barkana
Anadolu University
48 PUBLICATIONS **882** CITATIONS
SEE PROFILE

Soft Comput (2010) 14:365–378
DOI 10.1007/s00500-009-0410-8

ORIGINAL PAPER

Speeding up the scaled conjugate gradient algorithm and its application in neuro-fuzzy classifier training

Bayram Cetişli · Atalay Barkana

Published online: 11 March 2009

Abstract The aim of this study is to speed up the scaled conjugate gradient (SCG) algorithm by shortening the training time per iteration. The SCG algorithm, which is a supervised learning algorithm for network-based methods, is generally used to solve large-scale problems. It is well known that SCG computes the second-order information from the two first-order gradients of the parameters by using all the training datasets. In this case, the computation cost of the SCG algorithm per iteration is more expensive for large-scale problems. In this study, one of the first-order gradients is estimated from the previously calculated gradients without using the training dataset. To estimate this gradient, a least square error estimator is applied. The estimation complexity of the gradient is much smaller than the computation complexity of the gradient for large-scale problems, because the gradient estimation is independent of the size of dataset. The proposed algorithm is applied to the neuro-fuzzy classifier and the neural network training. The theoretical basis for the algorithm is provided, and its performance is illustrated by its application to several examples in which it is compared with several training algorithms and well-known datasets. The empirical results indicate that the proposed algorithm is quicker per iteration time than the SCG. The algorithm decreases the training time by 20–50% compared to SCG; moreover, the convergence rate of the proposed algorithm is similar to SCG.

B. Cetişli (✉)
Computer Engineering Department, Suleyman Demirel University, Bati Campus, 32260 Isparta, Turkey
e-mail: bcetisli@mmf.sdu.edu.tr

A. Barkana
Electrical and Electronics Engineering Department, Anadolu University, Iki Eylul Campus, 26470 Eskisehir, Turkey

Keywords Speeding up learning · Gradient estimation · The scaled conjugate gradient algorithm · Neuro-fuzzy classifier · Neural network · Large-scale problems

1 Introduction

Most of the recognition and classification problems consist of medium- and large-scale datasets, such as genetic research, character- or face recognition. Many different methods, such as neural networks (NNs), support vector machines, and Bayes classifier, have been proposed to solve these problems. The network-based methods can be trained with gradient based methods, and the calculation of new points of the network parameters generally depend on the size of the datasets. When the sizes of datasets are increased, the calculation time of the parameters for any iteration also increase. In this case, the training time of networks is a main problem for large-scale parameters.

One of the network-based classifiers is the neuro-fuzzy classifier (NFC), which combines the powerful description of fuzzy classification techniques with the learning capabilities of NNs. Generally, first-order gradient algorithms are used for the training of NFCs (Jang et al. 1997; Keles et al. 2007; Sun and Jang 1993). NFCs are widely used for different problems (Keles et al. 2007; Sinha and Fieguth 2006; Toosi and Kahani 2007). However, NFCs are not suitable for large-scale problems because of their many nonlinear network parameters. As a result, the training of NFCs with personal computers is quite slow and practically impossible. For these reasons, one of the aims of this study is to show that NFCs are able to classify large-scale classification problems successfully. Towards this aim, the scaled conjugate-gradient (SCG) algorithm is applied in NFC training because of its convergence rate and memory

usage. Ribeiro et al. (2006) used a modified version of SCG for training a type-1 fuzzy system. They indicated that the modified SCG speeds up the convergence in the fuzzy-system training according to the steepest-descent method. As a result, the SCG seems suitable for NFC training to solve large-scale problems. However, the training NFC with SCG for large-scale problems could consume more time, such as days or weeks, with any personal computer.

To decrease the training time, various methods have been developed. These methods can be grouped into two categories: heuristic and numerical. Some of the heuristic methods are based on the gradient-descent method. In these methods, the momentum term, the adaptive learning rate, and the resilient algorithm are used to accelerate convergence (Bishop 1996; Demuth et al. 2008; Moghaddam and Matinfar 2007; Mukkamala et al. 2005; Schraudolph 2002). Some algorithms based on linear least-square methods have been proposed to initialise or train feed-forward NNs (Castillo et al. 2006). Initialising the network parameters accelerates convergence (Chuang and Jeng 2007). The other type of heuristic method is to rescale the elements of the Jacobian matrix of the network parameters (Steil 2006). Some of the numerical methods, such as the conjugate gradient, quasi-Newton, and Levenberg–Marquardt methods, have been developed to speed up convergence (Bishop 1996; Broyden 1967; Levenberg 1944; Marquardt 1963; Møller 1993, 1997; Shanno 1970).

In general, the supervised learning algorithms can also be analysed using gradients. A popular gradient based algorithm is the steepest descent method, which uses the first-order gradient. Its form and its calculations are simple; however, it has a poor convergence rate. Therefore, the first-order gradient based algorithms are not suitable for large-scale and complex problems. Some alternative methods have been developed by using second-order information for increasing the convergence rate (Bishop 1996; Jang et al. 1997).

A well-known method is the Newton's method, which uses the second-order gradients of the parameters. All the other second-order numerical methods are based on the Newton's method. The convergence rate of Newton's method is higher than the rate of the first-order gradient algorithms. Nevertheless, using Newton's method in large-scale problems adds some difficulties: the definition of the Hessian matrix, the calculation of its inverse matrix, the heavy computational cost associated with the Hessian matrix, and the extensive memory usage. To overcome these difficulties and to speed up the learning process, the Hessian matrix or its inverse can be estimated (Bazaraa et al. 2006; Jang et al. 1997; Moghaddam and Matinfar 2007). Another alternative approach is the calculation of the Hessian matrix by using the first-order gradients as in conjugate gradient (CG) algorithms (Jang et al. 1997; Møller 1993). The CG algorithms use the line-search methods to determine the step size. A third approach is to calculate the Hessian matrix directly from the eigenvectors corresponding to its largest eigenvalues (Le Cun et al. 1991).

The SCG is based on the second-order gradient supervised learning procedure (Møller 1993). The SCG executes a trust-region step instead of the line-search step to scale the step size. The line search requires more parameters to determine the step size, and it increases the training time for any learning method. In a trust-region method, the distance for which the model function will be trusted is updated at each step. If the model step lies within that distance, it is used; in other cases, an approximate minimum for the model function on the boundary of the trust region is used. The trust-region methods are more robust than line-search methods (Møller 1993, 1997). The disadvantage of the line-search method is eliminated in the SCG by using the trust-region method as in the Levenberg–Marquardt method (Møller 1993).

In any iteration, however, the SCG computes two first-order gradients for the parameters to determine the second-order information. Therefore, the training time of any iteration in the SCG is two times higher than the steepest descent method (Møller 1993, 1997). Consequently, when the SCG is directly used for NFC training, it results in a slowing down for large-scale problems. The problem of training time is also a current problem for other network-based and numerical methods. Nevertheless, Møller's algorithm has been applied widely in solving large-scale problems (Abraham 2004; Kashiyama et al. 2000; Mukkamala et al. 2005; Sözen et al. 2005; Tran et al. 2004).

However, the increased learning time of the SCG for any iteration is still a problem. This disadvantage can be eliminated with gradient estimation. When the SCG is used to minimise the cost function of the networks, the gradients should be calculated over all the training samples. As the number of samples is increased, the time required for the training also increases. Two gradients are calculated per iteration of the SCG: the first gradient is calculated with a small step size, and the second gradient is calculated with a bigger step size. The first gradient of the cost function can be estimated rather than calculated because the estimation error of the first gradient will always be smaller than the estimation error of the second gradient because of the step sizes. These estimations shorten the training time of large-scale problems because they are independent of the number of samples. Furthermore, the estimation of gradients can smooth the cost function. In this manner, some of the local minimum points of cost functions can be eliminated. In this study, a least square estimator (LSE) is used to reduce the number of operations and the processing time of the SCG.

LSE is a simple method and requires less computation than other estimation or approximation methods in literature (Bishop 1996; Jang 1993; Jang et al. 1997; Theoridis and Koutroumbas 2003). Moreover, gradient estimation has rarely been used previously in the training phase (Le Cun et al. 1989; Edmonson et al. 1998; Wang and Principe 1999).

The proposed new algorithm will be referred to as "speeding up scaled conjugate gradient algorithm" (SSCG). The SSCG algorithm shortens the training time per iteration of the SCG; however, it does not affect the convergence rate of the training. To test the effectiveness of the algorithm, it is applied to the training of NFCs and NNs. Although SSCG is used for NN training also, the structure of the NN and its mathematical representations are not presented in this article. The performances of both NFC and NN are evaluated in the experimental part of this study. Empirical studies on speeding up the SCG show that the SSCG algorithm does not change the overall performance of convergence of the classifiers compared with that using the SCG.

To evaluate the usefulness of the SSCG, the NFCs and NNs, which are trained with SSCG, are used in six classification problems: thyroid disorders, letter recognition, adult statistics, connect-4 openings, applications to nurseries, and modified National Institute of Standards and Technology (MNIST) handwritten-digit recognition databases. For these problems, the SSCG algorithm is compared to the conventional SCG; resilient back-propagation (RP); the Broyden, Fletcher, Goldfarb and Shanno (BFGS) quasi-Newton; and the Powell–Beale conjugate gradient (PBCG) algorithms in terms of classification rate and training times (Broyden 1967; Jang et al. 1997; Shanno 1970). The RP algorithm is a first-order gradient based method, and the signs, instead of the magnitudes, of the gradients are used to determine the step size. Consequently, it is faster than the second-order methods. The PBCG and the BFGS quasi-Newton algorithms are second-order algorithms. The BFGS quasi-Newton algorithm is slower than the other algorithms because of the requirement for calculation of Hessian matrix (Jang et al. 1997; Møller 1993).

This article is organised as follows: in Sect. 2, the NFC and its training with the speeding up SCG algorithm are provided. The experimental results and comparison of the SSCG with the traditional SCG and other algorithms are detailed in Sect. 3. In Sect. 4, these results are discussed. Finally, Sect. 5 contains some concluding remarks.

2 NFC training with the speeding-up SCG algorithm

In the neuro-fuzzy classification methods, the feature space is partitioned into multiple fuzzy subspaces that are controlled by fuzzy if-then rules. These rules can be represented by a network structure. However, to determine an optimum fuzzy region, the parameters of the fuzzy rules should be optimised (Jang 1993; Jang et al. 1997; Sun and Jang 1993). The NFC consists of the following layers: fuzzy membership, fuzzification, defuzzification, normalisation and output. This classifier has both multiple inputs and multiple outputs. The use of weights in the defuzzification layer affects the rules and improves the classification flexibility. The K-means clustering method is used to obtain the initial parameters and to formulate the fuzzy if-then rules (Bishop 1996; Jang et al. 1997; Theoridis and Koutroumbas 2003).

A fuzzy classification rule R_i, which describes the relation between the input feature space and the classes, can be defined as follows:

$$R_i : \text{If } x_{p1} \text{ is } \Phi_{i1} \text{ and}\ldots\text{and } x_{pj} \text{ is } \Phi_{ij} \text{ and}\ldots\text{and } x_{pn} \text{ is } \Phi_{in} \text{ then class is } out_k,$$

where x_{pj} denotes the jth feature or input variable of the pth sample; out_k represents the kth label of class; n represents the number of features; Φ_{ij} denotes the fuzzy set of the jth feature in the ith rule and is characterised by the appropriate membership function (Sun and Jang 1993).

An NFC that is similar to Jang's classifier is shown in Fig. 1. The feature space in Fig. 1 has two features $\{x_1, x_2\}$ and the classifier separates them into three classes $\{out_1, out_2, out_3\}$. The network structure of the NFC is shown in the study by Sun and Jang (1993). Every feature is defined with three linguistic terms. As a result, there are nine fuzzy rules.

The parameters of NFC $\boldsymbol{\theta} = \{\Gamma_{m\times n}, \Lambda_{m\times n}, \mathbf{W}_{m\times c}\}$ can be adapted by supervised training algorithms, where $\boldsymbol{\Gamma}$ and $\boldsymbol{\Lambda}$ are the centre and the width matrices of the Gaussian

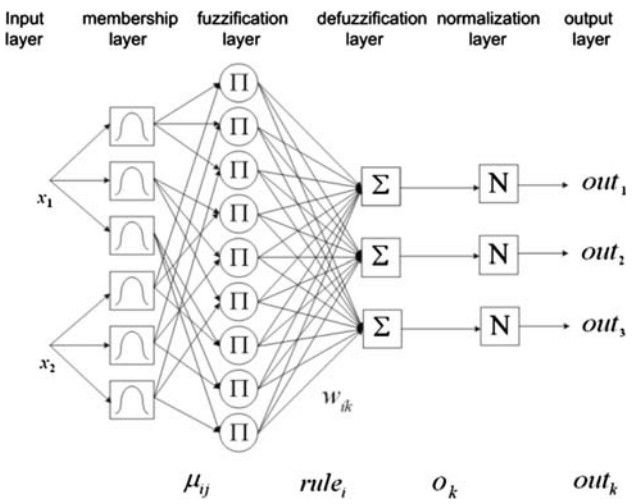

Fig. 1 A neuro-fuzzy classifier

membership function, respectively; $\mathbf{W}$ represents the weight matrix among the rules and the classes; m, n and c are the number of rules, the number of features and the number of classes, respectively. The adaptive neuro-fuzzy networks have also been trained using different optimisation methods, such as the Kalman filter, the standard back-propagation algorithm, and the Levenberg–Marquardt method (Haykin 2001; Jang 1991, 1993; Jang and Mizutani 1996; Jang et al. 1997). In this study, the SCG and SSCG algorithms have been used to optimise the parameters of the NFC.

Detailed step-by-step descriptions of the SCG algorithm can be found in the study by Møller (1993). However, some important definitions in the SCG algorithm should be mentioned to explain the reasons for the increased training time per iteration. The aim of the SCG algorithm is to determine the optimal NFC parameters $\boldsymbol{\theta}^*$ from the cost function $E(\boldsymbol{\theta})$.

When the cost function $E(\boldsymbol{\theta}_k)$ of the kth iteration is defined using the second-order truncated Taylor-series approximation, to find the search direction and the step size, the next point $\boldsymbol{\theta}_{k+1}$, which is close to the current point $\boldsymbol{\theta}_k$, is determined as below:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{g}_k \mathbf{H}_k^{-1}, \tag{1}$$

where $\mathbf{g}_k = E'(\boldsymbol{\theta}_k)$ and $\mathbf{H}_k = E''(\boldsymbol{\theta}_k)$ are, respectively, the gradient vector and the Hessian matrix of $E(\boldsymbol{\theta}_k)$. In Eq. 1, the step $-\mathbf{H}_k^{-1}\mathbf{g}_k$ is called the Newton step and its direction is called the Newton direction (Jang et al. 1997). If the Hessian matrix is positive-definite and $E(\boldsymbol{\theta}_{k+1})$ is quadratic, then the Newton's method directly arrives at a local minimum in a single Newton step (Jang et al. 1997). In other instances, reaching the local minimum requires more iteration. The calculation of the Newton step could be impossible for large-scale problems. Hence, Møller defined a temporal point $\boldsymbol{\theta}_{t,k}$, which is between $\boldsymbol{\theta}_{k+1}$ and $\boldsymbol{\theta}_k$, as follows:

$$\boldsymbol{\theta}_{t,k} = \boldsymbol{\theta}_k + \gamma_k \mathbf{d}_k, \quad \text{where} \quad 0 < \gamma_k \ll 1, \tag{2}$$

γ_k is the step size and $\mathbf{d}_k$ is the conjugate direction of the kth parameter.

Thus, the second-order information $-\mathbf{H}_k^{-1}\mathbf{g}_k$ is computed by using the following first-order information and the conjugate directions:

$$s_k = E''(\boldsymbol{\theta}_k)\mathbf{d}_k \approx \frac{E'(\boldsymbol{\theta}_{t,k}) - E'(\boldsymbol{\theta}_k)}{\gamma_k}, \quad \alpha_k = \frac{-\mathbf{d}_k^T E'_q(\boldsymbol{\theta}_1)}{\mathbf{d}_k^T s_k}, \tag{3}$$

where $\mathbf{s}_k$ denotes the second-order information; α_k denotes the basic and long step size. To compute α_k, the second-order information $\mathbf{s}_k$ should be obtained from the first-order gradients (Møller 1993). The next point is calculated as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \mathbf{d}_k \tag{4}$$

It can be observed from the above definitions that there are three main operations in the SCG algorithm. One of them is the evaluation of $E(\boldsymbol{\theta}_k)$; others are the calculation of $E'(\boldsymbol{\theta}_k)$ and $E'(\boldsymbol{\theta}_{t,k})$ and they are computed across all the training samples. Below, how the operation size of the gradient calculations can be halved with the estimation of the gradient $E'(\boldsymbol{\theta}_{t,k})$ is shown.

In general, quadratic approximations with Taylor-series expansions are used to find the optimum point that locally minimises the cost function. These quadratic approximations require the first-order and the second-order derivatives of the parameters to minimise the cost function $E(\boldsymbol{\theta})$. There is a similarity between the approximation of the cost function $E(\boldsymbol{\theta})$ and the estimation of the gradient function $\mathbf{g}$ due to their quadratic forms. However, the authors of this study applied the quadratic estimator to find the gradient of the temporarily shifted point $\boldsymbol{\theta}_{t,k}$ in the kth iteration of the SCG. In the quadratic gradient estimation, the first-order and the second-order derivatives of the gradient function are not required because the parabola coefficients can be estimated using LSE. Figure 2 shows that any gradient function $g(\theta)$ of the cost function $E(\theta)$ can be locally estimated using parabolas in the SSCG algorithm, where θ is any given network parameter.

If the gradient function illustrated in Fig. 2 is investigated, it can be observed that there are concave, convex and linear parts. These parts can be defined within certain intervals using quadratic forms and they can be approximated with parabolas (Bazaraa et al. 2006; Le Cun et al. 1989; Jang et al. 1997). When the parabolas fit the gradient function very well in an SCG step, similar to the

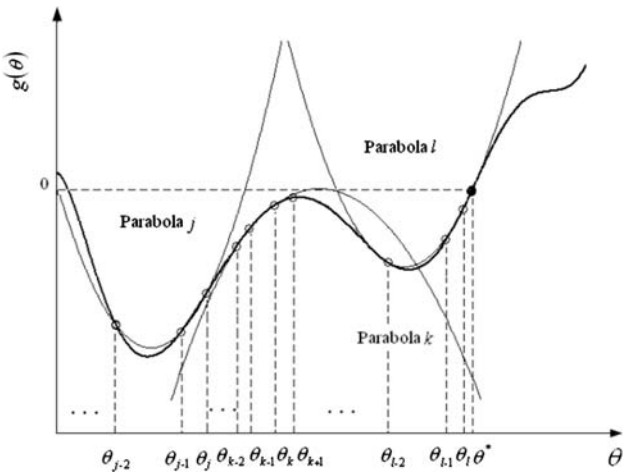

Fig. 2 The local estimations of the gradient function $g(\theta)$ of the cost function $E(\theta)$ using parabolas in the SSCG method

Parabola j and the Parabola l in Fig. 2, the gradient of the next point can be correctly estimated. When the parabola cannot approximate the gradient function, as with the Parabola k in Fig. 2, the step size can be misestimated due to the large estimation error. Therefore, the working area for the quadratic forms should be limited or partitioned to subregions, or the gradient of the next point to be estimated must be very close to the current point. When the errors in the parameters of the estimation process are cumulative, the estimated gradients may diverge from the real values because the objective function can immediately switch from concave to convex, or vice versa, within a narrow region. Hence, the gradient estimation is locally determined around the current θ_k parameter.

In the SCG algorithm, two gradients are calculated in any iteration: the gradient of the temporary point, which is calculated with the short step size, and the gradient of the real point, which is calculated with the long step size. For the temporary point, the deviation of the gradients is very small, i.e. $\left|g_{t,k}^{calc} - g_k\right| < \varepsilon$, where ε is a small constant number, because θ_k is much closer to $\theta_{t,k}$ than to θ_{k+1}. Hence, the error in gradient estimation also becomes small. After clarifying this point, it can be claimed that the gradient of the temporarily shifted point $\theta_{t,k}$ in the kth iteration is more suitable for estimation than the gradient of the point θ_{k+1}. In Fig. 3, this suitability is illustrated with the Parabola k. This figure also shows the calculated and the estimated gradients of the points, which are used in the kth iteration.

First of all, to explore the reason for estimating the gradient of the temporary point only, the calculations of θ, which were derived before in Eqs. 2 and 4, should be reanalysed for two different step sizes for any iteration:

$$\theta_{t,k} = \theta_k + \gamma_k d_k, \quad \theta_{k+1} = \theta_k + \alpha_k d_k, \tag{5}$$

where γ_k represents the normalised short step and is very close to the zero value, α_k represents the long step size and $\theta_{t,k}$ and θ_{k+1} are the parameters calculated by using two different step sizes. In Eq. 5, the step $\gamma_k d_k$ is smaller than the step $\alpha_k d_k$ because $\gamma_k \ll \alpha_k$. The steps for the kth iteration are shown in Fig. 3. When the cost functions $E(\theta)$ of any classifier are optimised by the SCG algorithm, the values of γ_k and α_k obtained throughout the training are shown in Fig. 4.

As observed in Fig. 4, the values of the step size γ_k are much smaller than the step size values of α_k. If the gradient estimation method is to be used in the SCG algorithm, the gradient estimation of the step γ_k is more suitable than the gradient estimation of the step α_k because of the possibility of a smaller estimation error, as observed in Fig. 3. Hence, the gradient estimation of each step γ_k cannot be much different from the actual gradient value. However, the gradient estimation of each step α_k yields a large estimation error owing to the longer step size of α_k. The estimation of one gradient only shortens the iteration time substantially.

In this study, the gradient functions are approximated by parabolas; the coefficients of parabolas are obtained by the LSE method. This method requires at least three points on the function in a quadratic form Jang et al. (1997). Therefore, to estimate these gradients with the parabolas, three points previously calculated on the gradient function are necessary (Thomas and Finney 1995). The calculations of the parabola coefficient **F** are shown below:

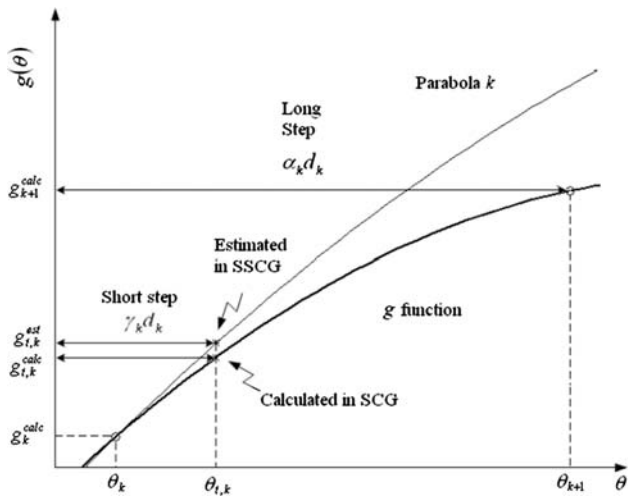

Fig. 3 The presentation of the calculated and the estimated gradients in the kth iteration for the SCG and the SSCG methods together

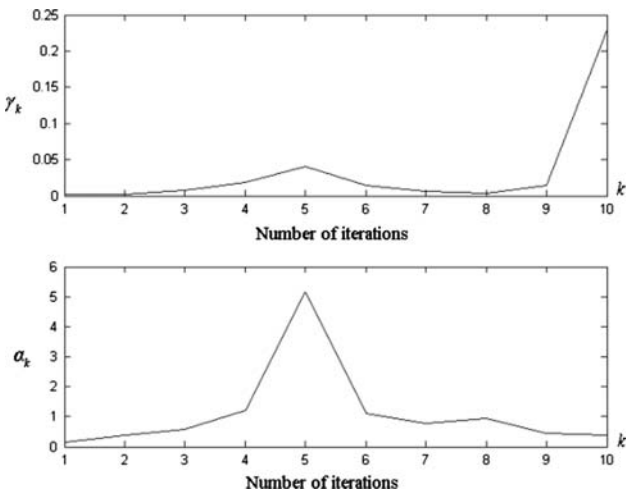

Fig. 4 The change of the step sizes γ_k and α_k for ten iterations during the training the cost function $E(\theta)$ with the SCG algorithm

$\mathbf{AF} = \mathbf{G} \Rightarrow \mathbf{F} = \mathbf{A}^{-1}\mathbf{G}$, where

$$\mathbf{A} = \begin{bmatrix} \theta_{k-2}^2 & \theta_{k-2} & 1 \\ \theta_{k-1}^2 & \theta_{k-1} & 1 \\ \theta_k^2 & \theta_k & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{\Theta}_{k-2} \\ \mathbf{\Theta}_{k-1} \\ \mathbf{\Theta}_k \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \quad \text{and} \quad \mathbf{G} = \begin{bmatrix} g_{k-2} \\ g_{k-1} \\ g_k \end{bmatrix}. \tag{6}$$

In Eq. 6, θ_k represents any parameter of the cost function in the kth iteration. The gradient $g_{t,k}^{est}$ of the temporarily shifted point $\theta_{t,k}$ can be estimated with $\mathbf{F}$

$$g_{t,k}^{est} = \mathbf{\Theta}_{t,k}\mathbf{F} = f_1\theta_{t,k}^2 + f_2\theta_{t,k} + f_3. \tag{7}$$

Three points and their gradients may not be sufficient to estimate $g_{t,k}^{est}$ in a few of the cases. If the gradient function in the subregion is not smooth enough, its oscillation due to additive noise along a main envelope cannot be named either as a convex or a concave function. Then, it can be deduced that it should be smoothed (Bazaraa et al. 2006). Additionally, if the values of the parameter in the latest three iterations θ_{k-2}, θ_{k-1}, θ_k are very close to each other, then the matrix $\mathbf{A}$ may be singular; the inverse of $\mathbf{A}$ may not exist. Using four or more points can eliminate these problems in the estimation of the parabola coefficient $\mathbf{F}$.

The gradient estimation using a priori four points is represented as

$$\mathbf{AF} = \mathbf{G} \Rightarrow \mathbf{F} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{G}. \tag{8}$$

Note that the gradient estimation in Eq. 8 is more costly than the computation in Eq. 6.

In addition, occasionally, there could be undesirable cases such as a zero bias term f_3, an infinite number, or a number that is not estimated for $g_{t,k}^{est}$ due to a large flat objective function area. In these cases, the estimated gradient cannot be used within the algorithm, and the gradient should be directly calculated. However, these negative conditions rarely occur in real-world problems, and using the K-means algorithm in NFC satisfies the condition that the training algorithm should start at a point nearby the local minimum.

At this point, speeding up of the SCG (SSCG) algorithm can be introduced. The SSCG algorithm is similar to Møller's SCG algorithm. The main difference between SSCG and SCG is the direction of the gradient estimation. Because the gradient estimation is independent of the number of samples, SSCG is faster than SCG. To estimate the initial gradient, three points and their gradients should be obtained before training. The SSCG algorithm is presented below.

Initialisation:

1. Set $k = 1$. Choose the initial values of the parameters $\boldsymbol{\theta}_{k-2}$ and $0 < \gamma < 1$. Then, calculate the gradient and the conjugate direction of $\boldsymbol{\theta}_{k-2}$:
   $\mathbf{d}_{k-2} = -\mathbf{g}_{k-2} = -E'(\boldsymbol{\theta}_{k-2})$, and set success = true.
2. Calculate the short step of the $(k-1)$th iteration operations:

$$\gamma_{k-1} = \frac{\gamma}{|\mathbf{d}_{k-2}|}, \quad \boldsymbol{\theta}_{k-1} = \boldsymbol{\theta}_{k-2} + \gamma_{k-1}\mathbf{d}_{k-2} \quad \text{and} \quad \mathbf{g}_{k-1} = E'(\boldsymbol{\theta}_{k-1}).$$

3. Calculate the long step size of the kth iteration; then find the new real point $\boldsymbol{\theta}_k$ and its gradient:
   $\mathbf{g}_k = E'(\boldsymbol{\theta}_k)$, $\mathbf{s}_k = \frac{E'(\boldsymbol{\theta}_{k-1}) - E'(\boldsymbol{\theta}_{k-2})}{\gamma_{k-1}}$, $\alpha_k = \frac{\mathbf{d}_k^T E'(\boldsymbol{\theta}_k)}{\mathbf{d}_k^T s_k}$, and $\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-2} + \alpha_k\mathbf{d}_{k-2}$. Set $\mathbf{\Theta}$, $\mathbf{A}$ and $\mathbf{G}$ using the results obtained above for the next gradient estimation:
   For $i = 1$ to M

$$\mathbf{A}_i = \begin{bmatrix} \mathbf{\Theta}_{k-2}(i) \\ \mathbf{\Theta}_{k-1}(i) \\ \mathbf{\Theta}_k(i) \end{bmatrix}_{3\times 3} = \begin{bmatrix} \theta_{k-2}^2(i) & \theta_{k-2}(i) & 1 \\ \theta_{k-1}^2(i) & \theta_{k-1}(i) & 1 \\ \theta_k^2(i) & \theta_k(i) & 1 \end{bmatrix}, \quad \text{and}$$

   $\mathbf{G}_i = [\, g_{k-2}(i) \quad g_{k-1}(i) \quad g_k(i) \,]^T$.
   End

The first three steps are necessary to estimate the new gradients and are carried out only once before running the SSCG algorithm. Therefore, the first three steps are not included in the repetition phase. The repetition phase of the SSCG algorithm is presented below.

Repetitions:

4. Set success as true, and

$$\mathbf{d}_k = -\mathbf{g}_k = -E'(\boldsymbol{\theta}_k).$$

5. The temporary point is determined and its gradient is estimated as follows:

$$\gamma_k = \frac{\gamma}{|\mathbf{d}_k|}, \quad \boldsymbol{\theta}_{t,k} = \boldsymbol{\theta}_k + \gamma_k\mathbf{d}_k.$$

For $i = 1$ to M

$$\mathbf{A}_i\mathbf{F}_i = \mathbf{G}_i.$$

If $\mathbf{A}_i$ is non-singular and has an inverse, then $\mathbf{F}_i = \mathbf{A}_i^{-1}\mathbf{G}_i$. Contrarily, if $\mathbf{A}_i$ is singular or has no inverse, then $\mathbf{F}_i = \left(\mathbf{A}_i^T\mathbf{A}_i\right)^{-1}\mathbf{A}_i^T\mathbf{G}_i$.
End

$$g_{t,k}^{est}(i) = E'\left(\theta_{t,k}(i)\right) = \begin{bmatrix} \left(\theta_{t,k}(i)\right)^2 & \theta_{t,k}(i) & 1 \end{bmatrix}^T \mathbf{F}_i.$$

If $g_{t,k}^{est}(i)$ is infinite or is not a number, then $g_{t,k}(i) = E'\left(\theta_{t,k}(i)\right)$ is directly calculated.
End
End

6. If success is true, then the second-order information is calculated as follows:

$$\mathbf{s}_k = \frac{\mathbf{g}_{t,k}^{est} - \mathbf{g}_k}{\gamma_k}, \quad \delta_k = \mathbf{d}_k^T \mathbf{s}_k.$$

7. Scaling $\mathbf{s}_k$:

$$\mathbf{s}_k = \mathbf{s}_k + \left(\lambda_k - \bar{\lambda}_k\right)\mathbf{d}_k, \quad \delta_k = \delta_k + \left(\lambda_k - \bar{\lambda}_k\right)|\mathbf{d}_k|^2.$$

8. If $\delta_k \leq 0$, then the Hessian matrix must be made positive-definite:

$$\mathbf{s}_k = \mathbf{s}_k + \left(\lambda_k - 2\frac{\delta_k}{|\mathbf{d}_k|^2}\right)\mathbf{d}_k, \quad \bar{\lambda}_k = 2\left(\lambda_k - \frac{\delta_k}{|\mathbf{d}_k|^2}\right),$$
$$\delta_k = -\delta_k + \lambda_k|\mathbf{d}_k|^2, \quad \lambda_k = \bar{\lambda}_k.$$

9. The calculations of the step size α_k and the new real point $\boldsymbol{\theta}_{k+1}$ are as follows:

$$\tau_k = \mathbf{d}_k^T \mathbf{g}_k, \quad \alpha_k = \frac{\tau_k}{\delta_k}, \quad \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \mathbf{d}_k.$$

10. The calculation of the reference for comparison:

$$\Delta_k = \frac{2\delta_k(E(\boldsymbol{\theta}_k) - E(\boldsymbol{\theta}_{k+1}))}{\tau_k^2}.$$

11. If $\Delta_k \geq 0$, then minimisation of the cost function is achieved:
$\mathbf{g}_{k+1} = E'(\boldsymbol{\theta}_{k+1})$, $\bar{\lambda}_k = 0$
and success = true.

- Add the new values $\boldsymbol{\theta}_{k+1}$ and $\mathbf{g}_{k+1}$ into $\mathbf{A}$ and $\mathbf{G}$ below, and then, the first rows of $\mathbf{A}$ and $\mathbf{G}$ are removed.
- If $(k \bmod M) = 0$, then the algorithm is restarted: $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1}$.

Or else, a new conjugate direction is created:

$$\beta_k = \frac{|\mathbf{g}_{k+1}|^2 - \mathbf{g}_{k+1}\mathbf{g}_k}{\tau_k}, \quad \mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k.$$

- If $\Delta_k \geq 0.75$, then the scaling parameter is decreased: $\lambda_k = 0.5\lambda_k$.

Otherwise, minimisation of the cost function is impossible: $\bar{\lambda}_k = \lambda_k$, and success = false.

12. If $\Delta_k < 0.25$, then the scaling parameter is increased: $\lambda_k = 4\lambda_k$.
13. If the steepest direction $\mathbf{g}_k \neq 0$, then set $k = k + 1$ and go to step 5; otherwise, the algorithm is completed and $\boldsymbol{\theta}_{k+1}$ is the desired minimum point of the cost function.

Hinton determined the complexity of calculating either $E(\boldsymbol{\theta}_k)$ or $\mathbf{g}_k$ as $O(M^2)$ over N training samples for a feedforward NN, where M represents the number of network parameters (Le Cun et al. 1989; Møller 1993, 1997). Gradient calculation for the SCG is repeated two times at any iteration. These are the calculations of $\mathbf{g}_{t,k}^{calc}$ and $\mathbf{g}_{k+1}$. In addition, there is one call of $E(\boldsymbol{\theta}_k)$ for each iteration. Consequently, Møller determined the calculation complexity of the SCG algorithm for any iteration as $O(3M^2)$ (Møller 1993). In this study, however, the first-order gradient $\mathbf{g}_{t,k}^{calc}$ is estimated instead of being calculated. The complexity of estimation of the gradient $\mathbf{g}_{t,k}^{est}$ is much smaller than that of calculating the gradient $\mathbf{g}_{t,k}^{calc}$ for large-scale problems because the gradient estimation for any parameter is independent of the data size. The explanation for this decrease in the number of operations is provided in the Appendix. Thus, the computational complexity $O(3M^2)$ can be decreased to $O(2M^2)$ in the SSCG algorithm for medium- and large-scale problems. The gradient estimation decreases the training time; however, it does not cause any performance degradation for the classifier. The proposed gradient-estimation method is independent of the number of samples, features and classes. These are the reasons why the speeded-up SCG algorithm works faster than SCG.

3 Experimental studies

The proposed SSCG and the SCG algorithms are applied to the training of NFCs and NNs. In the experimental studies, three medium-scale and three large-scale classification problems are used. Furthermore, to check the performance of the SSCG, resilient back-propagation, the BFGS quasi-Newton, and the PBCG algorithms are used in only NN training for comparison purposes. In the experimental studies, the classification of thyroid disease, letter recognition, adult statistics, connect-4 openings, applications to nurseries, and modified NIST handwritten-digit recognition datasets have been focussed on UCI Machine Learning Group http://www.ics.uci.edu/∼mlearn and http://yann.lecun.com/exdb/mnist/.

In the empirical studies, the networks are run five times for each problem, and the classification rates and training times are obtained for 100 epochs only. Due to this reason, the current classification results may be slightly lower or higher than other results in the literature. However, a comparison of the classification methods is not provided in this article, because it is out of the scope of this study. The purpose of this study is to speed up the SCG algorithm while conserving the convergence rate. The experimental studies are conducted on a personal computer with a Dual Core 1.83 GHz processor, 1 GB RAM memory, Windows XP and MATLAB 6.5 environment.

3.1 Thyroid disease dataset

The thyroid classification problem is a medium-scale problem because of the size of the dataset. There are 7,200 samples in the dataset. The initial 3,772 samples are used in the training set and the remaining 3,428 samples are used in the test set. There are three classes in the dataset and each sample has 21 features. Some of the features are eliminated because of their non-Gaussian distributions for some classes and only 11 features are applied in the classification (UCI Machine Learning Group http://www.ics.uci.edu/~mlearn). Table 1 lists the classification results and the training times.

3.2 Letter recognition dataset

There are 20,000 samples in the dataset. The initial 16,000 samples and the remaining 4,000 samples are used in the training and the test sets, respectively. This is a medium-scale classification problem. In the dataset, there are 26 classes and each sample has 16 features (UCI Machine Learning Group http://www.ics.uci.edu/~mlearn). The classification results and the training times are presented in Table 2.

3.3 Adult statistics dataset

The adult statistics dataset was extracted from the US Census Bureau's Income dataset. Each record has features that characterise an individual's annual income together with a class label indicating whether the person earned more or less than 50,000 dollars per year. The dataset has 48,842 samples and each sample has 14 features (6 continuous and 8 nominal) (UCI Machine Learning Group http://www.ics.uci.edu/~mlearn). After removing the data vectors with missing attribute values, the first 30,162 samples are considered for the training dataset; the final 15,060 samples are considered as the test dataset. The classification results and the training times are shown in Table 3.

3.4 Connect-4 opening dataset

This dataset contains all the legal eight player positions in the game of connect-4, in which neither player has won yet and in which the next move is not forced. There are 67,557 samples. The first 45,038 samples, which are randomly selected, and the remaining 22,519 samples are used in the training and test datasets, respectively. In the dataset, there are three classes and each sample has 42 nominal features (UCI Machine Learning Group http://www.ics.uci.edu/~mlearn). The classification results and the training times are provided in Table 4.

3.5 Nursery-application dataset

A nursery-application dataset was derived from a hierarchical decision model originally developed to rank applications for nursery schools. The final decision

Table 1 Classification results of the thyroid disease dataset

Classifier	Training algorithm	Training set R. (%)	Test set R. (%)	Unit iteration time (s)	Ratio	Shorten training time (%)	Cluster size per class
NFC	SCG	92.59	92.31	0.21	2.06	52.9	2
	SSCG	93.82	93.43	0.10	1.00		
	SCG	91.46	91.04	0.39	3.81	46.5	4
	SSCG	91.84	91.00	0.20	1.99		
	SCG	92.23	91.75	0.95	9.21	38.9	10
	SSCG	93.32	92.83	0.58	5.57		
NN	SCG	94.40	93.64	0.13	1.88	43.5	20-5 hidden neurons
	SSCG	96.89	94.80	0.079	1.07		
	RP	97.20	96.25	0.074	1.00	–	
	PBCG	93.02	93.82	0.22	3.12	–	
	BFGS	93.55	92.38	0.27	3.67	–	

Table 2 Classification results of the letter recognition dataset

Classifier	Training algorithm	Training set R. (%)	Test set R. (%)	Unit iteration time (s)	Ratio	Shorten training time (%)	Cluster size per class
NFC	SCG	78.02	75.94	15.17	1.39	28.5	2
	SSCG	79.27	77.31	10.85	1.00		
	SCG	86.30	84.18	37.18	3.42	48.8	4
	SSCG	86.02	84.00	20.52	1.89		
	SCG	90.74	88.70	118.10	10.87	39.3	10
	SSCG	90.97	88.84	71.57	6.59		
NN	SCG	90.37	85.05	4.02	1.97	41.6	104-52 hidden neurons
	SSCG	89.12	84.68	2.34	1.17		
	RP	87.88	84.40	2.04	1.00	–	
	PBCG	88.02	84.36	4.97	2.43	–	
	BFGS	–	–	–	–	–	

Table 3 Classification results of the adult statistics dataset

Classifier	Training algorithm	Training set R. (%)	Test set R. (%)	Unit iteration time (s)	Ratio	Shorten training time (%)	Cluster size per class
NFC	SCG	83.80	83.70	1.92	1.66	40.10	2
	SSCG	84.12	83.93	1.15	1.00		
	SCG	83.71	83.59	4.25	3.69	40.94	4
	SSCG	83.33	83.38	2.51	2.18		
	SCG	83.77	83.47	6.52	5.66	41.10	10
	SSCG	83.79	83.45	3.84	3.33		
NN	SCG	79.28	79.13	1.11	2.09	51.35	20-5 hidden neurons
	SSCG	78.45	77.93	0.53	1.00		
	RP	78.69	75.78	0.58	1.09	–	
	PBCG	75.25	74.39	2.88	5.43	–	
	BFGS	82.85	82.55	2.38	4.49	–	

Table 4 Classification results of the connect-4 opening dataset

Classifier	Training algorithm	Training set R. (%)	Test set R. (%)	Unit iteration time (s)	Ratio	Shorten training time (%)	Cluster size per class
NFC	SCG	72.21	71.89	0.13	1.85	46.15	2
	SSCG	72.05	71.63	0.07	1.00		
	SCG	73.40	73.03	0.28	4.00	46.42	4
	SSCG	73.05	72.63	0.15	2.14		
	SCG	74.47	74.24	0.42	6.00	45.23	10
	SSCG	74.29	74.16	0.23	3.28		
NN	SCG	73.76	73.27	0.097	1.79	44.32	80-20 hidden neurons
	SSCG	72.96	72.82	0.054	1.00		
	RP	68.16	67.87	0.066	1.22	–	
	PBCG	75.25	74.39	0.14	2.59	–	
	BFGS	–	–	–	–	–	

depended on three subcriteria: occupation of parents and child's nursery, family structure and financial standing, and the social and health picture of the family. In the dataset, there are five classes (recommended, not recommended, much recommended, priority and special priority), and each sample has eight features (UCI Machine Learning Group

http://www.ics.uci.edu/∼mlearn). There are 12,960 samples. In this study, the recommended class, which has only two samples, is removed because it has insufficient number of samples. The remaining samples are equally divided between the training and the test sets. The classification results and the training times are presented in Table 5.

3.6 Modified NIST handwritten-digit dataset

The MNIST dataset, which includes 60,000 training images and 10,000 testing images, was extracted from the NIST special datasets SD3 and SD7. All the images are in binary format and their size is normalised to fit in a 20 × 20 pixel box (http://yann.lecun.com/exdb/mnist/). Within the dataset, the normalised images are located in 28 × 28 planes and centred using their centre of mass. The classification results and the training times are provided in Table 6.

4 Discussion

Regarding the behaviour of the SSCG algorithm as a training algorithm, experiments are conducted and the results are presented in Sect. 3. There are five main features of the SSCG algorithm that are prominent:

1. Speeding up the SCG algorithm: Experimental studies show that the proposed speeding-up proposal for the SCG is successful for the NFC and the NN training. When the complexity of the classification problem and the number of samples of datasets are increased, the difference in the training times at 20–50% rates between the SCG and the SSCG is increased in favour of the SSCG. It implies that the SSCG algorithm is faster than the SCG algorithm considering the training times.

Table 5 Classification results of the nursery-application dataset

Classifier	Training algorithm	Training set R. (%)	Test set R. (%)	Unit iteration time (s)	Ratio	Shorten training time (%)	Cluster size per class
NFC	SCG	95.13	93.31	0.30	1.66	40.5	2
	SSCG	92.94	91.54	0.18	1.00		
	SCG	94.28	91.83	0.54	3.00	48.1	4
	SSCG	95.26	91.94	0.28	1.55		
	SCG	96.68	92.12	1.95	10.87	39.4	10
	SSCG	96.49	92.51	1.18	6.59		
NN	SCG	97.45	91.26	0.79	1.88	43.3	80-20 hidden neurons
	SSCG	94.72	90.47	0.45	1.07		
	RP	96.48	91.23	0.42	1.00	–	
	PBCG	97.34	91.20	1.20	2.85	–	
	BFGS	95.70	90.63	7.66	18.28	–	

Table 6 Classification results of the MNIST dataset

Classifier	Training algorithm	Training set R. (%)	Test set R. (%)	Unit iteration time (s)	Ratio	Shorten training time (%)	Cluster size per class
NFC	SCG	89.41	88.25	118.20	1.85	46.10	2
	SSCG	89.26	87.92	63.70	1.00		
	SCG	91.28	90.07	246.05	3.86	44.20	4
	SSCG	90.89	90.14	137.28	2.15		
	SCG	96.58	95.27	589.14	9.24	42.73	10
	SSCG	96.71	94.83	337.40	5.29		
NN	SCG	95.66	94.19	275.43	1.69	40.89	600-40 hidden neurons
	SSCG	95.72	94.37	162.80	1.00		
	RP	95.63	94.12	238.49	1.46	–	
	PBCG	–	–	–	–	–	
	BFGS	–	–	–	–	–	

2. Maintaining the convergence rate: When the experimental results are analysed, the SSCG yields similar convergence rates as the SCG. On certain occasions, the SSCG is better than the SCG, or vice versa, considering their convergence rates. The reason for this switching is the estimation error in the gradients.
3. Initialising the network parameters: In the NFC, K-means clustering is used to initialise the antecedent parameters of NFC and to construct the fuzzy rules. This operation increases the convergence rate and is satisfied to avoid some of the local minima. However, a similar situation is not defined for NN. As a result, NFC-recognition results can be better than NN-recognition results, as observed from the experimental studies.
4. Using the NFC for large-scale problems: The classification of the MNIST handwritten dataset, which is a large-scale dataset, is a benchmark recognition problem. Nevertheless, fuzzy-based classifiers have not been used for this problem because they need more fuzzy rules and more nonlinear network parameters. In these situations, the use of fuzzy-based classifiers for large-scale problems is impossible with personal computers. In this study, however, the classification of the MNIST dataset is easily executed. Nevertheless, the recognition rate using NFC for the MNIST dataset may not be better than other classification methods (Zhang et al. 2007). The reason is that no feature extraction or selection method is used in this study.
5. Comparison of the training methods: In this study, RP, PBCG and BFGS algorithms are also tested as classification methods for every problem. The RP and the PBCG algorithms are suitable for medium- and large-scale problems (Demuth et al. 2008). Generally, the RP algorithm is the best among all the compared methods according to the training time. The second-best algorithm is the SSCG. In addition, the experimental studies show that the SCG is better than the PBCG. The difference is due to the use or otherwise of the line-search method. The BFGS is the worst algorithm due to its requirement for more memory storage. Therefore, in a few problems, the BFGS-recognition results could not be obtained due to insufficient memory. When the recognition rates are compared, the BFGS algorithm is generally better than the other algorithms. The second-best algorithms are both SCG and SSCG algorithms. The worst is the RP algorithm. Moreover, NFCs are generally slower than NNs because the NFCs have nonlinear parameters and have more complex architecture. However, the recognition performance of NFC is generally better than that of the NN with the same number of network parameters. According to the experimental results, it can be concluded that the SCG and the proposed SSCG algorithms are more suitable for network-based classifiers for large-scale classification problems.

5 Conclusions

It is hereby shown that one can use the quadratic approximation for gradient estimation in the second-order approximation of the cost function.

The training time of the second-order supervised learning algorithms may be shortened by using gradient estimation. Gradient estimation does not excessively affect the performance of the supervised learning algorithms. Generally, the calculations of gradients depend on the number of samples, features and classes; however, they can be estimated independent of the training set. This goal can be achieved by using the gradient estimation method. When the NFCs are trained with supervised learning algorithms, the derivative chains are directly proportional to the number of layers. Accordingly, their computational complexities increase with the number of layers in the network. However, in the proposed estimation method, the gradients calculated from the samples in the previous iterations are used to estimate the new values of the gradients in the next iteration.

The reason for using SCG for gradient estimation in NFC training is that SCG calculates two different gradients of the parameters in each iteration. The gradient of the temporarily shifted point $\theta_{t,k}$ in the kth iteration is more suitable than the gradient of the alternate point θ_{k+1} for gradient estimation. In the SSCG algorithm, the SCG algorithm is improved by using gradient estimation and the second gradient is used to estimate the first gradient for the next iteration. In the SSCG algorithm, the computational complexity of SCG is decreased from $O(3M^2)$ to $O(2M^2)$. One disadvantage of SSCG is that it requires three values of the previously computed gradient information.

The LSE method has been used in this study. However, other estimation methods, such as the Kalman Filter, the Wiener Filter and the Expectation–Maximisation method can also be used.

The SSCG algorithm has been applied to NFCs and NNs in this study. The improved algorithm can also be used in other network based and second-order gradient based training methods.

Acknowledgments The authors thank Rifat Edizkan, Omer Nezih Gerek, and the reviewers for all the useful discussions and their valuable comments on this article.

Appendix: The presentation of the complexity of the gradient estimation

Our claim is that the complexity of the gradient estimation $g_{t,k}^{est}$ is less than the complexity of the gradient calculation $g_{t,k}^{calc}$, that is $O\left(g_{t,k}^{est}\right) < O\left(g_{t,k}^{calc}\right)$, where $O(\cdot)$ is the calculation complexity.

This can be proved with ease:

Both $g_{t,k}^{est}$ and $g_{t,k}^{calc}$ represent the gradient of E with respect to ρ_{ij}.

Calculation of the operation size of $g_{t,k}^{est}$;

$$\text{If } \mathbf{A} = \begin{bmatrix} \theta_{k-2}^2 & \theta_{k-2} & 1 \\ \theta_{k-1}^2 & \theta_{k-1} & 1 \\ \theta_k^2 & \theta_k & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{\Theta}_{k-2} \\ \mathbf{\Theta}_{k-1} \\ \mathbf{\Theta}_k \end{bmatrix},$$
$$\text{then } O(\mathbf{A}) = 3\,mult \tag{9}$$

In Eq. 9, the "*mult*" term represents the operation of multiplication.

$$\text{If } \mathbf{F} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \quad \text{and}$$
$$\mathbf{G} = \begin{bmatrix} g_{k-2} \\ g_{k-1} \\ g_k \end{bmatrix} \Rightarrow \mathbf{AF} = \mathbf{G} \Rightarrow \mathbf{F} = \mathbf{A}^{-1}\mathbf{G}. \tag{10}$$

$g_{t,k}^{est}$ is estimated by LSE as shown below:

$$g_{t,k}^{est} = \mathbf{\Theta}_{t,k}\mathbf{F} = f_1\theta_{t,k}^2 + f_2\theta_{t,k} + f_3. \tag{11}$$

If $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$, then the determinant of $\mathbf{A}$ is calculated as follows:

$$|\mathbf{A}| = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31}. \tag{12}$$

In addition $O(|\mathbf{A}|) = 5\,add + 2 \times 6\,mult$, where the term "*add*" represents the operation of the addition. The adjoint of $\mathbf{A}$ is given below:

$$adj(\mathbf{A}) = \begin{cases} \mathbf{A}_{11} = (-1)^{1+1} = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} = a_{22}a_{33} - a_{23}a_{32} \\ \vdots \\ \mathbf{A}_{33} = (-1)^{1+1} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12} \end{cases}. \tag{13}$$

The operation size of $adj(\mathbf{A})$ is $O(adj(\mathbf{A})) = 9\times(1\,add + 2\,mult)$. Now, $\mathbf{A}^{-1}$ and its size can be determined as;

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} adj(\mathbf{A}) = \mathbf{B}, \quad \text{and}$$
$$O\left(\mathbf{A}^{-1}\right) = 14\,add + 39\,mult. \tag{14}$$

$$\mathbf{F} = \mathbf{A}^{-1}\mathbf{G} = \begin{bmatrix} b_{11}g_{k-2} + b_{12}g_{k-1} + b_{13}g_k \\ b_{21}g_{k-2} + b_{22}g_{k-1} + b_{23}g_k \\ b_{31}g_{k-2} + b_{32}g_{k-1} + b_{33}g_k \end{bmatrix} \quad \text{and}$$
$$O(\mathbf{F}) = 6\,add + 9\,mult. \tag{15}$$

Lastly, $O\left(g_{t,k}^{est} = \mathbf{\Theta}_{t,k}\mathbf{F}\right) = 2\,add + 2\,mult$ and $Total\ O = 22\,add + 53\,mult$.

On the other hand, the calculation of $g_{t,k}^{calc}$ and its number of arithmetic operations are determined below:

Firstly, the output of NFC should be calculated from the first layer of NFC:

$$\mu_{ij} = \exp\left(-0.5\frac{\left(x_{pj} - \rho_{ij}\right)^2}{\sigma_{ij}^2}\right), \tag{16}$$

where $\rho_{ij}\ (\rho_{ij} \in \mathbf{\Gamma}_{m\times n})$ and $\sigma_{ij}\ (\sigma_{ij} \in \mathbf{\Lambda}_{m\times n})$ are the centre and the width of the Gaussian membership function μ_{ij} for the ith rule and the jth feature. In Eq. 16, the $\exp(\cdot)$ operation is approximately calculated using the Maclaurin Series;

$$\exp(z) = \sum_{u=0}^{\infty}\frac{z^u}{u!} = 1 + z + \frac{z^2}{2!} + \cdots + \frac{z^u}{u!} + \cdots \tag{17}$$

If the upper limit for $u = 10$, then $O(\exp(z)) = 10\,add + 117\,mult$ and $O\left(\mu_{ij}\right) = 11\,add + 123\,mult$. The output of the second layer is calculated as

$$rule_i = \prod_{j=1}^{n}\mu_{ij}, \quad \text{and}$$
$$O(rule_i) = n \times (11\,add + 123\,mult), \tag{18}$$

where $rule_i$ represents the firing strength of the ith rule. The output of the third layer is calculated as:

$$o_k = \sum_{i=1}^{m} rule_i w_{ik}, \quad \text{and}$$
$$O(o_k) = m \times n \times (11\,add + 123\,mult) + m \times (add + mult), \tag{19}$$

where $w_{ik}\ (w_{ik} \in \mathbf{W}_{m\times c})$ represents the weight of the ith rule that belongs to the kth class; m denotes the number of rules. The output of the last layer is calculated as

$$out_k = \frac{o_k}{\sum_{l=1}^{c} o_l} = \frac{o_k}{\mathbf{T}}, \quad \mathbf{T} = \sum_{l=1}^{c} o_l \quad \text{and}$$
$$O(out_k) = m \times n \times (11\,add + 123\,mult) + (m + c)\,add + (m + 1)\,mult. \tag{20}$$

The mean square error function is used as the cost function:

$$E = \frac{1}{N}\sum_{p=1}^{N} E^p, \quad E^p = \frac{1}{2}\sum_{k=1}^{c}\left(y_{pk} - out_{pk}\right)^2, \tag{21}$$

where N represents the number of samples; c represents the number of classes; y_{pk} and out_{pk} are the target and the actual output of the pth sample and the kth class, respectively.$g_{t,k}^{calc}$ of ρ_{ij} is calculated using the chain rule:

$$\frac{\partial E}{\partial \rho_{ij}} = \frac{1}{N}\sum_{p=1}^{N}\sum_{k=1}^{c}\left(out_{pk} - y_{pk}\right)\left(\frac{1 - out_{pk}}{\mathrm{T}}\right) w_{ik} rule_i \left(\frac{x_{pj} - \rho_{ij}}{\sigma_{ij}^2}\right). \tag{22}$$

The total arithmetic operations of $g_{t,k}^{calc}$ of ρ_{ij} are given below:

$$\begin{aligned} O\left(g_{t,k}^{calc}\right) &= c \times N \times (m \times n \times (11\,add + 123\,mult) \\ &\quad + (m + c)\,add + (m + 1)\,mult) \\ &\quad + c \times N \times (9\,mult + 3\,add) + 2(c \times N)\,add \\ &\quad + N \times n \times (11\,add + 123\,mult) \\ O\left(g_{t,k}^{calc}\right) &\approx (c \times N \times (5 + 11 \times m \times n) \\ &\quad + 11 \times N \times n)\,add \\ &\quad + (c \times N \times (9 + 11 \times m \times n) + 123 \times N \times n)\,mult \end{aligned} \tag{23}$$

The comparison of the number of arithmetic operations of the gradients is given in Eq. 24.

$$\begin{aligned} &22\,add + 53\,mult \ll N \times n \times \left(\left(c \times \left(\frac{5}{n} + 11 \times m\right) + 11\right)\right. \\ &\left.\times add + \left(c \times \left(\frac{9}{n} + 11 \times m\right) + 123\right)\right) mult \end{aligned}$$

If $N > 100$, $m > 1$, $n > 1$ and $c > 1$, then

$$O\left(g_{t,k}^{est}\right) \ll O\left(g_{t,k}^{calc}\right) \tag{24}$$

References

Abraham A (2004) Meta learning evolutionary artificial neural networks. Neurocomputing 56:1–38. doi:10.1016/S0925-2312(03)00369-2

Bazaraa MS, Sherali HD, Shetty CM (2006) Nonlinear programming, 3rd edn. Wiley, New York

Bishop CM (1996) Neural networks for pattern recognition. Oxford University Press, New York

Broyden CG (1967) Quasi-Newton methods and their applications to function minimization. Math Comput 21:368–381. doi:10.2307/2003239

Castillo E, Guijarro-Berdiñas B, Fontenla-Romero O, Alonso-Betanzos A (2006) A very fast learning method for neural networks based on sensitivity analysis. J Mach Learn Res 7:1159–1182

Chuang CC, Jeng JT (2007) CPBUM neural networks for modeling with outliers and noise. Appl Soft Comput 7:957–967. doi:10.1016/j.asoc.2006.04.009

Demuth H, Beale M, Hagan M (2008) Neural network toolbox 6 user's guide. Mathworks Inc, Natick

Edmonson W, Principe J, Srinivasan K, Wang C (1998) A global least mean square algorithm for adaptive IIR filtering. IEEE trans. on circuits and systems-II. Analog Digit Signal Process 45(3):379–384. doi:10.1109/82.664244

Haykin S (2001) Kalman filtering and neural networks. Wiley, New York

Jang JSR (1991) Fuzzy modelling using generalized neural networks and Kalman filter algorithm. In: Proceedings of the ninth national conference on artificial intelligence (AAAI-91), pp 762–767

Jang JSR (1993) ANFIS: adaptive network based fuzzy inference systems. IEEE Trans Syst Man Cybern 23:665–685. doi:10.1109/21.256541

Jang JSR, Mizutani E (1996) Levenberg-Marquardt method for ANFIS learning. In: Proceedings of the international joint conference of the north American fuzzy information processing society biannual conference, Berkeley, pp 87–91

Jang JSR, Sun CT, Mizutani E (1997) Neuro-fuzzy and soft computing. Prentice Hall, Upper Saddle River

Kashiyama K, Tamai T, Inomata W, Yamaguchi S (2000) A parallel finite element method for incompressible Navier-Stokes flows based on unstructured grids. Comput Methods Appl Mech Eng 190(3–4):333–344

Keles A, Hasiloglu AS, Keles A, Aksoy Y (2007) Neuro-fuzzy classification of prostate cancer using NEFCLASS-J. Comput Biol Med 37:1617–1628. doi:10.1016/j.compbiomed.2007.03.006

Le Cun Y, Galland C, Hinton GE (1989) GEMINI: gradient estimation through matrix inversion after noise injection. In: Touretzky D (ed) Advances in neural information processing systems 1 (NIPS'88). Morgan Kaufman, Denver

Le Cun Y, Kanter I, Solla SA (1991) Eigenvalues of covariance matrices: application to neural network learning. Phys Rev Lett 66(18):2396–2399. doi:10.1103/PhysRevLett.66.2396

Levenberg K (1944) A method for the solution of certain problems in least squares. Q Appl Math 2:164–168

Marquardt DW (1963) An algorithm for least squares estimation of nonlinear parameters. J Soc Ind Appl Math 11:431–441. doi:10.1137/0111030

Moghaddam HA, Matinfar M (2007) Fast adaptive LDA using quasi-Newton algorithm. Pattern Recognit Lett 28:613–621. doi:10.1016/j.patrec.2006.10.011

Møller M (1993) A scaled conjugate gradient algorithm for fast supervised learning. Neural Netw 6(4):525–533. doi:10.1016/S0893-6080(05)80056-5

Møller M (1997) Efficient training of feed-forward neural networks. Ph.D. Thesis, Aarhus University, Denmark

Mukkamala S, Sung AH, Abraham A (2005) Intrusion detection using an ensemble of intelligent paradigms. J Netw Comput Appl 28(2):167–182. doi:10.1016/j.jnca.2004.01.003

Ribeiro MV, Duque CA, Romano JMT (2006) An interconnected type-1 fuzzy algorithm for impulsive noise cancellation in multicarrier-based power line communication systems. IEEE J Sel Areas Communitications 24(7):1364–1376. doi:10.1109/JSAC.2006.874417

Schraudolph NN (2002) Fast curvature matrix-vector products for second-order gradient descent. Neural Comput 14:1723–1738. doi:10.1162/08997660260028683

Shanno DF (1970) Conditioning of quasi-Newton methods for function minimization. Math Comput 24:647–656. doi:10.2307/2004840

Sinha SK, Fieguth PW (2006) Neuro-fuzzy network for the classification of buried pipe defects. Autom Constr 15:73–83. doi:10.1016/j.autcon.2005.02.005

Sözen A, Arcaklioğlu E, Özalp M, Kanit EG (2005) Solar-energy potential in Turkey. Appl Energy 8(4):367–381. doi:10.1016/j.apenergy.2004.06.001

Steil JJ (2006) Online stability of backpropagation–decorrelation recurrent learning. Neurocomputing 69:642–650. doi:10.1016/j.neucom.2005.12.012

Sun CT, Jang JSR (1993) A neuro-fuzzy classifier and its applications. In: Proceedings of IEEE international conference on fuzzy systems, San Francisco, vol 1, pp 94–98

Theoridis S, Koutroumbas K (2003) Pattern recognition, 2nd edn. Academic Press, London

Thomas GB, Finney RL (1995) Calculus and analytic geometry, 9th edn. Addison-Wesley, Reading

Toosi NA, Kahani M (2007) A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. Comput Commun 30:2201–2212. doi:10.1016/j.comcom.2007.05.002

Tran C, Abraham A, Jain L (2004) Decision support systems using hybrid neurocomputing. Neurocomputing 61:85–97. doi:10.1016/j.neucom.2004.03.006

Wang C, Principe J (1999) Training neural networks with additive noise in the desired signal. IEEE Trans Neural Netw 10(6):1511–1517. doi:10.1109/72.809097

Zhang P, Bui TD, Suen CY (2007) A novel cascade ensemble classifier system with a high recognition performance on handwritten digits. Pattern Recognit 40:3415–3429. doi:10.1016/j.patcog.2007.03.022