Efficient First Order Superlinear Algorithms

Peter GÉCZY[1], Shotaro AKAHO[1], Shiro USUI[2]

[1]AIST NRI, Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki 305–8568, Japan
[2]RIKEN BSI, 2-1 Hirosawa, Wako-shi, Saitama 351-0198, Japan

Abstract. Substantial number of problems in artificial intelligence requires optimization. Increasing complexity of the problems imposes several challenges on optimization algorithms. The algorithms must be fast, computationally efficient, and scalable. Balance between convergence speed and computational complexity is of central importance. Typical example is the task of training neural networks. Superlinear algorithms are highly regarded for their speed-complexity ratio. With superlinear convergence rates and linear computational complexity they are often the primary choice. Two first order superlinear algorithms are introduced. The algorithms are computationally efficient, convergent, and theoretically justified. They are applied to several neural network training tasks, practically evaluated, and compared to the relevant first order optimization techniques. Results indicate superior performance.

Key words: first order optimization, superlinear convergence raters, steepest descent, conjugate gradient, line search, neural networks, learning algorithms.

1 Introduction

The computational complexity of first order methods scales linearly with the problem size as opposed to quadratic scaling of second order methods. First order methods provide first order convergence rates and zero or linear memory requirements. The second order methods have quadratic both convergence rates and memory requirements. Although quadratic convergence rates are faster than first order ones, the computational complexity substantially outweighs the rates in terms of processing time on conventional computing architectures. The fastest first order methods are superlinear. They are suitable for large scale optimization [1], [2].

Early theoretical results reported that the steepest descent with exact line search is globally convergent to a stationary point. This motivated the approaches aimed at choosing the step length $\alpha^{(k)}$ close to the exact line search values. Computational excess of these methods led to weakening the exact line search to the iterative decrease of the objective function, $E^{(k)} > E^{(k+1)}$. Only the iterative descent property is insufficient since it allows negligible reductions of E when the learning rate $\alpha^{(k)}$ approaches zero values, $\alpha^{(k)} \to 0$, and/or the search direction $s^{(k)}$ is nearly perpendicular to the gradient vector $\nabla E^{(k)}$. To avoid these occurrences, two conditions on $\alpha^{(k)}$ have been proposed in the early optimization literature [3];

$$E(u^{(k)} + \alpha^{(k)} s^{(k)}) \leq E(u^{(k)}) + \alpha^{(k)} \sigma \nabla E(u^{(k)}) \ , \tag{1}$$

$$E(u^{(k)} + \alpha^{(k)} s^{(k)}) \geq E(u^{(k)}) + \alpha^{(k)} (1-\sigma) \nabla E(u^{(k)}) \ , \tag{2}$$

where $\sigma \in (0, 0.5)$ is a fixed parameter. Conditions (1) and (2) work suitably well for quadratic functions. In the case of non-quadratic E, the condition (2) may exclude the minimizing point of $E(u^{(k)} + \alpha^{(k)} s^{(k)})$, hence it has been suggested to use the following condition instead (see [4]);

$$E(u^{(k)} + \alpha^{(k)} s^{(k)}) \geq \delta \nabla E(u^{(k)}) \ , \tag{3}$$

where $\delta \in (\sigma, 1)$ is a fixed parameter. This leads to more complicated convergence theorems [5]. A stronger condition, replacing (3), has also been considered (see [6]);

$$\left| E(u^{(k)} + \alpha^{(k)} s^{(k)}) \right| \leq -\delta \nabla E(u^{(k)}) \ . \tag{4}$$

Polynomial interpolations to the individual residuals rather than to the overall objective function have been observed to speed up the line search subproblem in nonlinear least squares [6]. This trend has been extensively

explored in machine learning. Initial approaches used heuristics to simplify the line search subproblem to a single step calculation of the learning rate and/or momentum term [7]. Heuristics introduced in [8] propose adapting a learning rate for a given weight u_i by $c\alpha_i^{(k)}$, $c > 1$, if $sgn(\frac{\partial E(u^{(k)})}{u_i}) = sgn(\frac{\partial E(u^{(k-1)})}{u_i})$; and $b\alpha_i^{(k)}$, $b \in (0,1)$, if $sgn(\frac{\partial E(u^{(k)})}{u_i}) \neq sgn(\frac{\partial E(u^{(k-1)})}{u_i})$. Another heuristic method suggests exponential rather than linear increases/decreases and no progression when oscillations occur [9]. Theoretically justified method for modifying the learning rate was presented in [10]: set $\alpha^{(k)} = \alpha^{(k-1)}$ if $sgn(\nabla E(u^{(k)})) = sgn(\nabla E(u^{(k-1)}))$ and $\alpha^{(k)} = 0.5\alpha^{(k-1)}$ otherwise.

Stochastic gradient descent algorithms perform sub-gradient parameter adaptation for each pair $[x, y] \in T$ (or subset of T) [11]. To satisfy the asymptotic convergence condition various adaptation strategies have been introduced: $\alpha^{(k)} = 1/k$ (see [12]), or $\alpha^{(k)} = \alpha^{(0)}/(1+k)$ (see [13]). Inverse proportionality of the learning rate $\alpha^{(k)}$ to a number of iterations k implies $\alpha^{(k)} \to 0$ when k grows large. This results in negligible reductions of the objective function and slow convergence. Increasing the initial $\alpha^{(0)}$ causes instability for small k. A variation addressing these issues has been reported: $\alpha^{(k)} = \alpha^{(0)}/[1 + (k/\tau)]$ (see [14]).

Conjugate gradient methods form the search direction by conjugating the scaled gradient with the scaled previous search direction [15]. This partially helps stabilizing the unwanted oscillatory behavior of the steepest descent methods. Several previous search directions can be employed [16], however, this increases memory requirements. Adaptability of the scaling factor of the previous search direction–the momentum term–improves the convergence speed. In early optimization literature several adaptation strategies have been recommended [17], [18]. Application of conjugate gradient methods to machine learning tasks produced additional heuristic adaptation strategies of momentum term [19].

1.1 Problem Formulation: Mapping and Training in MLP Networks

We start with rigorous formulation of mapping and training in three-layer MLP networks. They have been proven to have universal approximation capabilities [20], i.e. an arbitrary functional dependency can be approximated to an arbitrary level of accuracy by three-layer network with an appropriate number of nonlinear hidden units. Sigmoidal type of nonlinearity is a popular choice [21].

Definition 1 *(Mapping of a Three-Layer MLP Network)*
A mapping $\mathcal{F}$ is said to be a mapping of a three-layer MLP network defined as follows.

$$\mathcal{F} = \mathcal{F}_{HO} \circ \mathcal{F}_{IH} \quad (\mathcal{F} : \Re^{N_I} \to \Re^{N_O}) ,$$

where N_I is the dimensionality of the input space and N_O is the dimensionality of the output space. $\mathcal{F}_{HO}$ is an affine mapping from N_H-dimensional subspace $\mathcal{D}^{N_H}$ of $\Re^{N_H}$ to $\Re^{N_O}$.

$$\mathcal{F}_{HO} = (\mathcal{F}_{HO_1}, \ldots, \mathcal{F}_{HO_{N_O}}) \quad ; \quad \mathcal{F}_{HO} : \mathcal{D}^{N_H} \to \Re^{N_O}$$
$$\mathcal{F}^{(p)}_{HO_k} = \sum_{j=1}^{N_H} w_{jk} \mathcal{F}^{(p)}_{IH_j} - \theta_{o_k}$$

where $\mathcal{F}^{(p)}_{IH_k}$ is the output of the k-th hidden unit for the p-th training pattern, θ_{o_k} is the threshold value ($\theta_{o_k} \in \mathcal{R}$) for the k-th output unit, w_{jk} is the real valued weight connection connecting the j-th hidden unit with the k-th output unit. $\mathcal{F}_{IH}$ is nonlinear multidimensional mapping

$$\mathcal{F}_{IH} = \mathbf{f} \circ \mathcal{A}_{IH} \quad (\mathcal{F}_{IH} : \Re^{N_I} \to \mathcal{D}^{N_H}) ,$$
$$\mathcal{F}^{(p)}_{IH_j} = f\left(\sum_{i=1}^{N_I} v_{ij} x_i^{(p)} - \theta_{h_j} \right)$$

where θ_{h_j} is the threshold value ($\theta_{h_j} \in \mathcal{R}$) for the j-th hidden unit, v_{ij} is the real valued weight connection connecting the i-th input unit with the j-th hidden unit, $x_i^{(p)}$ is the i-th coordinate of the p-th input vector $x^{(p)}$, $\mathbf{f}$ stands for a multidimensional nonlinear sigmoidal transformation in which each dimension of its N_H-dimensional domain vector is transformed by a sigmoidal transfer function f, ($\mathbf{f} : \Re^{N_H} \to \mathcal{D}^{N_H}$), $\mathcal{A}_{IH}$ is an input-to-hidden affine submapping $\mathcal{A}_{IH} : \Re^{N_I} \to \Re^{N_H}$. ■

Definition 2 *(Training in MLP Networks)*
Let T be a training set with cardinality N_P,

$$T = \{[x, y] | x \in \Re^{N_I} \wedge y \in \Re^{N_O}\} \quad , \quad |T| = N_P ,$$

where each pair $[x, y]$ contains the input vector x of the dimensionality N_I, and the expected output vector y of the dimensionality N_O. Let u denote a set of free system parameters of a network and the objective function E be defined as follows,

$$E(u, T) = C \cdot \sum_{p=1}^{N_P} \sum_{k=1}^{N_O} (\mathcal{F}_k(u, x^{(p)}) - y_k^{(p)})^2, \text{ where } C \text{ is a constant.} \tag{5}$$

Training in MLP networks is a process of minimization the objective function E,

$$\arg\min_u E(u, T) \ , \tag{6}$$

given a finite number of samples $[x, y] \in T$ drawn from an arbitrary sample distribution. ■

2 Superlinear First Order Algorithms

Optimization algorithms iteratively optimize the objective function. Starting from the initial point they generate a sequence of points $\{u^{(k)}\}_k$ in a parameter space of E that should converge to the point in a solution set. General optimization scheme consists of the following stages:

- **Initialization:** determines the starting point of optimization and sets parameters specific to the used algorithm.
- **Progression:** describes the iterative procedure for generating the sequence of points $\{u^{(k)}\}_k$.
- **Termination:** specifies when the algorithm stops — stopping criterion.

Progression is the central part of an algorithm. It specifies how the algorithm progresses from the point $u^{(k)}$ to the next point $u^{(k+1)}$ in the parameter space of the objective function. Line search methods move along the line given by the search direction vector. The algorithm has to find the appropriate length of the progress, so as to achieve the maximum decrease of the objective function (minimization case). Length of the progress is controlled by the scaling factors: step length and/or momentum. Determining the suitable values of step length and/or momentum in a single step considerably relaxes the computational complexity of the line search subproblem.

Derivation of the presented algorithms utilizes the first order classification framework introduced in [22]; particularly, the algorithm descriptor:

$$\mathcal{AD} = \limsup \frac{||\Delta u^{(k)}||_2 \cdot ||\nabla E(u^{(k)})||_2}{|E(u^*) - E(u^{(k)})|} \ .$$

Substituting for $\Delta u^{(k)}$ from the parameter update of a given optimization technique the expressions for iterative updates of step length $\alpha^{(k)}$ and/or momentum $\beta^{(k)}$ imply. This approach results in novel first order steepest descent and conjugate gradient methods: ALGORITHM 1 and ALGORITHM 2.

ALGORITHM 1 *(Steepest Descent)*

1. *Set the initial parameters: $u^{(0)}$, a, and $E(u^*)$.*
2. *Calculate $E(u^{(k)})$, evaluate the stopping criteria and then either terminate or proceed to the next step.*
3. *Calculate the gradient $\nabla E(u^{(k)})$.*
4. *Calculate $\alpha^{(k)}$,*

$$|\alpha^{(k)}| = a \cdot \frac{|E(u^*) - E(u^{(k)})|}{||\nabla E(u^{(k)})||_2^2} \ . \tag{7}$$

5. *Update the system parameters as follows.*

$$u^{(k+1)} = u^{(k)} - \alpha^{(k)} \cdot \nabla E(u^{(k)})$$

6. *Set $k = k + 1$ and proceed to Step 2.*

■

ALGORITHM 2 *(Conjugate Gradient)*

1. *Set the initial parameters:* $u^{(0)}$, a, *and* $E(u^*)$.
2. *Calculate* $E(u^{(k)})$, *evaluate the stopping criteria and then either terminate or proceed to the next step.*
3. *Calculate the gradient* $\nabla E(u^{(k)})$.
4. *Calculate* $\alpha^{(k)}$,

$$|\alpha^{(k)}| = a \cdot \frac{|E(u^{(k)})|}{||\nabla E(u^{(k)})||_2^2} \ , \tag{8}$$

and $\beta^{(k)}$,

$$|\beta^{(k)}| = \frac{a \cdot E(u^*)}{||s^{(k-1)}||_2 \cdot ||\nabla E(u^{(k)})||_2} \ . \tag{9}$$

5. *Update the system parameters as follows.*

$$u^{(k+1)} = u^{(k)} - \alpha^{(k)} \cdot \nabla E(u^{(k)}) + \beta^{(k)} \cdot s^{(k-1)} \ .$$

6. *Set* $k = k + 1$ *and proceed to Step 2.*

■

ALGORITHM 1 and ALGORITHM 2 have linear computational complexity $O(N_F)$, where N_F is a number of free parameters. ALGORITHM 1 is memoryless. ALGORITHM 2 has linear memory requirements $O(N_F)$ resulting from the necessity of storing the previous search direction $s^{(k-1)}$. Despite the simplicity of the line search subproblem, both ALGORITHM 1 and ALGORITHM 2 are convergent.

Theorem 1 *(Convergence and Convergence Rates)*
Let E *be an objective function defined on the attractor basin* $\mathcal{L}$ *of the point* u^* *with continuous first derivatives. A sequence of points* $\{u^{(k)}\}_k$ *generated by ALGORITHM 1 or ALGORITHM 2 from the initial point* $u^{(0)} \in \mathcal{L}$ *converges to the terminal attractor* u^* *(with respect to the negligible residual* $R_{n\geq 2}$*),* $\{u^{(k)}\}_k \to u^*$, *with superlinear convergence rates.* ■

ALGORITHM 1 and ALGORITHM 2 are substantially more powerful, in terms of convergence speed, than the standard steepest descent and conjugate gradient methods used in BP training techniques for MLP neural networks. Higher flexibility of the step length and/or momentum term helps to determine the search direction of the algorithms more appropriately, and considerably eliminates the unwanted oscillatory behavior. Both ALGORITHM 1 and ALGORITHM 2 are capable of optimizing an objective function E with superlinear convergence rates.

2.1 Local Minima Escape Capabilities

Dynamic update of step length and/or momentum term enables ALGORITHM 1 and 2 to escape from local minima. The escape mechanism is based on the dramatic parameter update when $||\nabla E(u^{(k)})||_2$ is close to zero and objective function E is not approaching the expected value. Details and illustrative example are offered in the following paragraphs.

First order necessary condition for extreme, whether local or global, is zero gradient vector $\nabla E = \mathbf{0}$, and thus also $||\nabla E||_2 = 0$. As the algorithm converges to the optimum point, $||\nabla E(u^{(k)})||_2$ converges to zero, $||\nabla E(u^{(k)})||_2 \to 0$. Recall the expression for adaptable step length $\alpha^{(k)}$ (7). It is inversely proportional to the squared l_2 norm of the gradient. Thus, when the algorithm approaches the minimum, $||\nabla E(u^{(k)})||_2^2$ approaches zero, $||\nabla E(u^{(k)})||_2^2 \to 0$. If error value $E(u^{(k)})$ is not approaching expected value $E(u^*)$, the expression in the numerator of (7) is nonzero, $|E(u^*) - E(u^{(k)})| \neq 0$. This leads to large values of $\alpha^{(k)}$ and dramatic parameter update even for small coordinate values of gradient $\nabla E(u^{(k)})$.

Similar phenomenon can be observed also in dynamics of adaptable momentum term $\beta^{(k)}$ (9), providing $E(u^*) \neq 0$ and $a \neq 0$. Convergence of $||\nabla E(u^{(k)})||_2$ to zero during the convergence to the local minimum, and nonzero numerator, $a \cdot E(u^*) \neq 0$, results in large values of $\beta^{(k)}$. Then the nonzero coordinates of the vector of the previous search direction, $s^{(k-1)}$, multiplied by large $\beta^{(k)}$, contribute to the dramatic parameter update.

The evidence of the local minima escape capabilities is illustratively demonstrated in Figure 1. ALGORITHM 1 was applied to minimizing function $E(x, y) = x^4 - x^3 + 17x^2 + 9y^2 + y + 102$. The function is quadratic with respect to y and of the 4th order with respect to x. Function $E(x, y)$ has two minima; one global

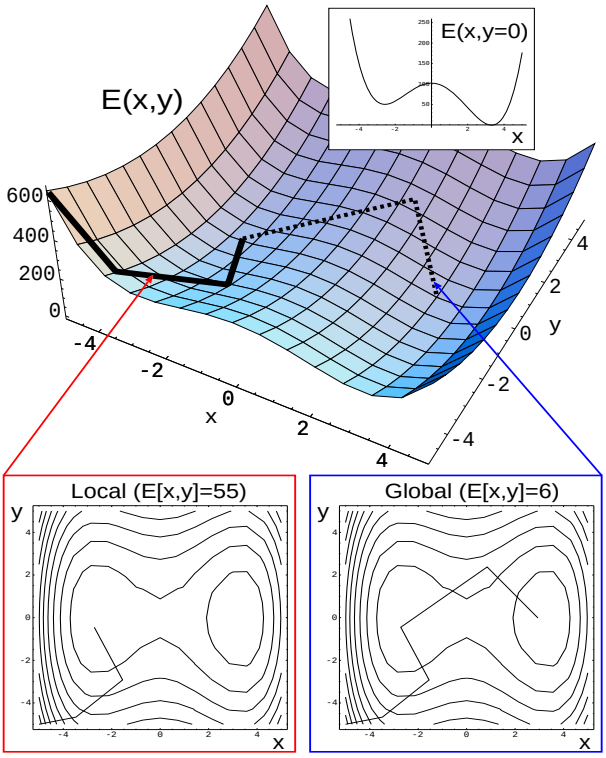

Figure 1: Demonstration of local minima escape capabilities.

and one local. Upper chart of Figure 1 shows the position of local and global minima at the cutting plane $y = 0$. The starting point of the optimization was $[x, y] = [-5, -5]$ and parameter a was set to 1. In the first case, the expected stopping function value was set to 55, $E(u^*) = 55$. Progress of ALGORITHM 1, given the above setting, is shown in bottom left contour plot of Figure 1. Since the expected functional value was the stopping criterion, the algorithm stopped after 4 iterations reaching the value $E(u^*) \leq 55$ in the local minimum. In the second case, the expected functional value was set to 6. The optimization progress of the algorithm is shown in bottom right contour plot of Figure 1. Starting from the point $[-5, -5]$, the algorithm initially converged to the local minimum. Small values of $||\nabla E(u^{(k)})||_2^2$ in the area around the local minimum, and discrepancy $|E(u^{(k)}) - E(u^*)|$, led to large values of $\alpha^{(k)}$ and dramatic parameter update. This caused the algorithm to jump out of the local minimum and finally (after escaping from the local minimum) in a single step to reach the appropriate value of E in the neighborhood of the global minimum.

3 Implementation Specifics

The presented algorithms incorporate the formulas for automatic adjustments of step length $\alpha^{(k)}$ and/or momentum term $\beta^{(k)}$ containing the choice of a. The parameter a directly relates to the algorithm class $\mathcal{AD}$. In the absence of the exact algorithm class value it is possible to determine a approximately. Following derivation relates to ALGORITHM 1.

Since ALGORITHM 1 is a modification of steepest descent technique it is reasonable to assume comparison of ALGORITHM 1 to the steepest descent algorithm in order to determine a. Without neglecting the second and higher order terms of Taylor expansion, $R_{n\geq 2}$, it follows for the standard steepest descent technique with constant α,

$$\left| 1 - \limsup |\alpha| \frac{||\nabla E(u^{(k)})||_2^2}{|E(u^*) - E(u^{(k)}) + R_{n\geq 2}|} \right| . \tag{10}$$

Then for ALGORITHM 1 the following is implied,

$$\left| 1 - \limsup \frac{a \cdot |E(u^*) - E(u^{(k)})|}{|E(u^*) - E(u^{(k)}) + R_{n\geq 2}|} \right| . \tag{11}$$

ALGORITHM 1 has better convergence rates than the standard steepest descent technique with constant step

length α if (11) $<$ (10), that is,

$$\begin{aligned}
\limsup \frac{a \cdot |E(u^*) - E(u^{(k)})|}{|E(u^*) - E(u^{(k)}) + R_{n\geq 2}|} &> \limsup |\alpha| \cdot \frac{||\nabla E(u^{(k)})||_2^2}{|E(u^*) - E(u^{(k)}) + R_{n\geq 2}|} \\
\frac{a \cdot |E(u^*) - E(u^{(k)})|}{|E(u^*) - E(u^{(k)}) + R_{n\geq 2}|} &> |\alpha| \cdot \frac{||\nabla E(u^{(k)})||_2^2}{|E(u^*) - E(u^{(k)}) + R_{n\geq 2}|} \\
a \cdot |E(u^*) - E(u^{(k)})| &> |\alpha| \cdot ||\nabla E(u^{(k)})||_2^2 \\
a &> |\alpha| \cdot \frac{||\nabla E(u^{(k)})||_2^2}{|E(u^*) - E(u^{(k)})|} \ . \qquad (12)
\end{aligned}$$

The expression (12) is the necessary condition for a when ALGORITHM 1 is used. Then from a short pre-training of the standard steepest descent with the constant step length α the parameter a can be determined as a value satisfying (12) by simply taking the maximum of the pre-training sequence.

$$\max_{k=1,\ldots,C_p} \left\{ |\alpha| \cdot \frac{||\nabla E(u^{(k)})||_2^2}{|E(u^*) - E(u^{(k)})|} \right\}_{k=1}^{C_p} \qquad (13)$$

Parameter C_p stands for a number of allowed pre-training cycles. Determination of the parameter a by (13) is naturally more precise when C_p grows large. Moving average, instead of the maximum, in (13) can also be used. Analogously, it is possible to monitor a for various values of constant step length.

Another pertinent issue related to the presented algorithms is a choice of the value $E(u^*)$. Seemingly obvious solution would be to assign $E(u^*)$ the value of the expected error. However, when the algorithm's stopping condition is a value of the expected error, then $\alpha^{(k)}$ converges to 0 as the algorithm approaches the expected error value. This results in slow convergence in the final phase of optimization. The simple solution avoiding this difficulty is setting $E(u^*)$ slightly lower than the value of the expected error. For certain classes of problems $E(u^*)$ can be determined differently. For example, in the least square problems (whether linear or nonlinear) the value of $E(u^*)$ can be determined from the boundness of the problem. Another solution is to implement adaptable strategy for $E(u^*)$.

The universality and superlinear convergence of the proposed algorithms rely on the appropriate setting of the parameters $E(u^*)$ and a. The exact values of the parameters in practice may be unknown. Though the above mentioned procedures can be applied to approximate the parameters, it should be noted that the theoretically obtained superlinear convergence no longer holds. However, even in such situation the algorithms should perform substantially well.

4 Simulations

The introduced algorithms were compared to the relevant techniques within the same class, that is, the first order methods and to the pseudo-second order method called Kick-Out [16]. Since both ALGORITHM 1 and ALGORITHM 2 are first order methods, it would be unreasonable to compare them to the second order ones. Kick-Out, however, utilizes the approximation of the second order information. It has been observed that Kick-Out outperforms conventional learning algorithms and their variations.

The effectiveness of the algorithms is practically demonstrated on five tasks represented by the following data sets: Lenses [23], Glass, Monks 1 [24], Monks 2 [24], and Monks 3 [24]. The presented algorithms were applied to training various MLP networks to perform tasks given by five, the above mentioned, data sets. Neural networks' performance was optimized according to the mean square error. Stopping criterion was the value of the expected error.

In the case of the Lenses data set [23], a neural network had configuration 4–3–1 with sigmoidal hidden units. Expected error was set to $5 \cdot 10^{-2}$. In the Glass problem, a network was configured as: 9–5–1 (sigmoidal hidden units) and the expected error was equal to 0.35. Finally, for Monks 1, 2, and 3 problems [24] a neural network structure was set as: 6–3–1 (sigmoidal hidden units), and the expected error was equal to 0.103. Network's weights were initialized randomly in the interval $< -0.1, 0.1 >$, which corresponded to the steepest region of the sigmoidal transfer function of the hidden units. ALGORITHM 1 and 2 used the value of $E(u^*) = 0$ (implying from the lower bound of the mean square error function E). The parameter a was equal to 1. In case network's error did not converge to the value less than or equal to the expected error within 20000 cycles, the training process was terminated. It is interesting to note that additional stopping condition of maximum 20000 cycles was practically applied only to the BP employing standard first order techniques. ALGORITHM 1 and 2 always converged.

Outline of the experiments follows. First, the standard BP with the constant step length term α ranging from 0.1 to 0.9 in 0.1 increments and Kick-Out algorithm were compared to ALGORITHM 1 (see Table 1).

Learning Rate α		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	**AVERAGE** Task	**AVERAGE** Total
Lenses	BP	11.1	5.58	3.68	3.58	4.05	5.28	3.99	3.96	1.79	**4.78**	
	KO	3.21	2.36	2.01	1.73	2.21	2.56	1.78	1.67	1.21	**2.08**	
Glass	BP	6.41	9.08	6.42	4.88	3.93	3.27	2.81	2.47	2.23	**4.61**	BP: **4.12**
	KO	2.35	3.78	3.21	2.1	1.58	1.32	1.22	1.19	1.11	**1.98**	
Monks 1	BP	7.69	7.51	5.7	4.49	3.67	3.08	2.49	2.22	2.61	**4.38**	
	KO	3.15	3.76	2.63	2.22	1.91	1.49	1.22	1.15	1.21	**2.08**	KO: **2.08**
Monks 2	BP	9.54	4.8	3.04	2.34	1.88	1.38	1.2	1.23	1.04	**2.93**	
	KO	4.1	3.12	2.72	2.52	1.23	1.19	0.98	1.1	1.12	**2.01**	
Monks 3	BP	7.2	6.58	4.37	3.2	2.66	2.78	2.9	3.71	1.64	**3.89**	
	KO	3.55	3.28	2.66	1.77	1.35	1.41	1.52	1.83	1.22	**2.07**	

Table 1: Comparison between ALGORITHM 1, standard BP and Kick-Out (KO) on several data sets. Elements of the table indicate how many times was ALGORITHM 1 faster than the standard BP and Kick-Out.

Momentum Term β		0.1	0.2	0.3	0.4	0.5	0.6	0.7	**AVERAGE** Task	**AVERAGE** Total
Lenses	BPM	2.05	1.26	1.11	1.88	14.66	20.62	34.84	**10.92**	
($\alpha = 0.9$)	KO	1.5	1.11	1.03	1.25	5.31	5.72	6.11	**3.15**	
Glass	BPM	2.24	2.14	1.96	1.85	1.70	1.61	1.43	**1.85**	
($\alpha = 0.9$)	KO	1.31	1.3	1.24	1.36	1.28	1.13	1.01	**1.23**	BPM: **7.38**
Monks 1	BPM	2.46	2.14	1.76	1.71	2.16	8.19	14.81	**4.75**	
($\alpha = 0.8$)	KO	1.32	1.26	1.21	1.17	1.33	2.21	3.15	**1.66**	KO: **2.18**
Monks 2	BPM	1.01	0.96	1.17	2.55	8.31	37.17	48.31	**14.21**	
($\alpha = 0.9$)	KO	0.97	0.96	1.11	1.35	3.24	5.89	6.73	**2.89**	
Monks 3	BPM	1.73	1.68	1.56	2.50	6.01	9.93	12.87	**5.18**	
($\alpha = 0.9$)	KO	1.25	1.21	1.18	1.77	2.31	2.66	3.24	**1.95**	

Table 2: Comparison of ALGORITHM 2, BPM and Kick-Out (KO) with learning rate setting corresponding to the best obtained former results against ALGORITHM 1.

Remaining experiments were performed with the value of the step length (learning rate) corresponding to the best results of BP and Kick-Out against ALGORITHM 1 (in Monks 1 case $\alpha = 0.8$, and for all other data sets $\alpha = 0.9$). The momentum term ranging from 0.1 to 0.7 in 0.1 increments was then added. Kick-Out and BP with the momentum term and the best value of step length (denoted in further text as BPM) were compared to ALGORITHM 2 (see results in Table 2). Kick-Out's additional parameters were set as follows: $\kappa = 0.0001$, $\phi = 0.9$ and $\tau = 0.7$. For a given setting of learning rate and momentum term the simulations were run 10 times for different randomly initialized weights in the interval $< -0.1, 0.1 >$. The values in Table 1 and 2 represent ten-run-averages. Convergence speed increase values in the tables indicate how many times the proposed algorithms converged faster than BP, BPM and Kick-Out techniques. Criterion for comparison of the convergence speed was the number of cycles required to decrease the mean square error E of a neural network below the value of the expected error.

It is clear, from Table 1 and 2 that the proposed algorithms converged substantially faster. ALGORITHM 1 was on the average over all five tasks approximately 4 times faster than BP and 2 times faster than Kick-Out. Performance of ALGORITHM 2 indicated approximately 7 times faster convergence than BPM and 2 times faster than Kick-Out. As previously mentioned, ALGORITHM 1 and 2 converged each time, whereas BP and BPM for some initial setting of weights and parameters α, β could not achieve convergence even after 20000 cycles.

5 Conclusions

Two superlinear algorithms are presented. Algorithms feature automatically adjustable step length $\alpha^{(k)}$ and/or momentum term $\beta^{(k)}$ in a single step calculation. The steepest descent ALGORITHM 1 adjusts only step length $\alpha^{(k)}$ at each iteration of optimization procedure. It is memoryless with linear computational complexity $O(N_F)$, where N_F is a number of free parameters in a system. The conjugate gradient ALGORITHM 2 dynamically adjusts step length $\alpha^{(k)}$ and momentum term $\beta^{(k)}$. Computational complexity and memory requirements

of ALGORITHM 2 are linear, $O(N_F)$. The proposed algorithms are capable of achieving the superlinear convergence rates. They are convergent, computationally inexpensive, easily implementable, and in practice very suitable for large scale optimization–whether in terms of data size or number of parameters. In cases where amount of available memory plays inevitable role, ALGORITHM 1 is advantageous since it is memoryless. Otherwise, due to the high flexibility of step length $\alpha^{(k)}$ and momentum term $\beta^{(k)}$, the preferred choice may be ALGORITHM 2.

Practical validation of the presented algorithms was performed on five data sets: Lenses, Glass, Monks 1, 2 and 3. The algorithms were compared to the relevant first order line search optimization techniques: the steepest descent, the conjugate gradient, and Kick-Out. Simulation results show satisfactory performance.

Acknowledgment

The authors would like to thank Dr. Shun-ichi Amari of RIKEN and Prof. Naohiro Toda of Aichi Prefectural University for their useful comments.

References

[1] Leon S. Lasdon. *Optimization Theory for Large Systems*. Dover, New York, 2002.

[2] H. Frenk, K. Roos, T. Terlaky, and S. Zhang (Editors). *High Performance Optimization*. Kluwer Academic Publishers, Dordrecht, 1999.

[3] A. A. Goldstein. On steepest descent. *SIAM Journal of Control*, 3:147–151, 1965.

[4] P. Wolfe. Convergent conditions for ascent methods. *SIAM Review*, 11:226–235, 1969.

[5] M. J. D. Powell. A view of unconstrained optimization. In L. C. W. Dixon, editor, *Optimization in Action*, London, 1976. Academic Press.

[6] M. Al-Baali and R. Fletcher. An efficient line search for nonlinear least squares. *Journal of Optimization Theory and Application*, 48(3):359–377, 1986.

[7] R. A. Jacobs. Increasing rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[8] T. P. Vogl, J. K. Manglis, A. K. Rigler, T. W. Zink, and D. L. Alkon. Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, 59:257–263, 1988.

[9] T. Tollenaere. SuperSAB: Fast adaptive back propagation with good scaling properties. *Neural Networks*, 3:561–573, 1990.

[10] Ch. G. Pflug. Non-asymptotic confidence bounds for stochastic approximation algorithms. *Mathematic*, 110:297–314, 1990.

[11] J. C. Spall. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Essex, 2003.

[12] S. Amari. Theory of adaptive pattern classifiers. *IEEE Transactions*, EC–16(3):299–307, 1967.

[13] L. Ljung. Analysis of recursive stochastic algorithms. *IEEE Transactions on Control*, AC–22(3):551–575, 1997.

[14] C. Darken and J. Moody. Note on learning rate schedules for stochastic optimization. In R. P. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Proceedings of the Neural Information Processing Systems 3 (Denver)*, pp. 832–838, San Mateo, 1991. Morgan Kaufmann.

[15] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Comput. Journal*, 7:149–154, 1964.

[16] K. Ochiai, N. Toda, and S. Usui. Kick-Out learning algorithm to reduce the oscillation of weights. *Neural Networks*, 7(5):797–807, 1994.

[17] J. W. Daniel. Convergence of the conjugate gradient method with computationally convenient modifications. *Numerical Mathematics*, 10:125–131, 1967.

[18] D. F. Shanno. Conjugate gradient methods with inexact searches. *Mathematics of Operations Research*, 3(3):244–256, 1978.

[19] S. J. Perantonis and D. A. Karras. An efficient constrained learning algorithm with momentum acceleration. *Neural Networks*, 8(2):237–249, 1995.

[20] K. Hornik. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[21] A. Menon, K. Mehrotra, C.K. Mohan, and S. Ranka. Characterization of a class of sigmoid functions with application to neural networks. *Neural Networks*, 6(5):819–835, 1996.

[22] P. Géczy and S. Usui. Novel first order optimization classification framework. *IEICE Transactions on Fundamentals*, E83-A(11):2312–2319, 2000.

[23] J. Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27:349–370, 1987.

[24] J. Wnek and R. S. Michalski. Comparing symbolic and subsymbolic learning: Three studies. In R. S. Michalski and G. Tecuci, editors, *Machine Learning: A Multistrategy Approach*, volume 4, San Mateo, 1993. Morgan Kaufmann.