Communicated by Raymond Watrous

Partial BFGS Update and Efficient Step-Length Calculation for Three-Layer Neural Networks

Kazumi Saito
Ryohei Nakano
NTT Communication Science Laboratories,
2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Second-order learning algorithms based on quasi-Newton methods have two problems. First, standard quasi-Newton methods are impractical for large-scale problems because they require N^2 storage space to maintain an approximation to an inverse Hessian matrix (N is the number of weights). Second, a line search to calculate a reasonably accurate step length is indispensable for these algorithms. In order to provide desirable performance, an efficient and reasonably accurate line search is needed.

To overcome these problems, we propose a new second-order learning algorithm. Descent direction is calculated on the basis of a partial Broydon-Fletcher-Goldfarb-Shanno (BFGS) update with $2Ns$ memory space ($s \ll N$), and a reasonably accurate step length is efficiently calculated as the minimal point of a second-order approximation to the objective function with respect to the step length. Our experiments, which use a parity problem and a speech synthesis problem, have shown that the proposed algorithm outperformed major learning algorithms. Moreover, it turned out that an efficient and accurate step-length calculation plays an important role for the convergence of quasi-Newton algorithms, and a partial BFGS update greatly saves storage space without losing the convergence performance.

1 Introduction

The backpropagation (BP) algorithm (Rumelhart *et al.* 1986) has been applied to various classes of problems, and its usefulness has been proved. However, even with a momentum term, this algorithm often requires a large number of iterations for convergence. Moreover, the user is required to determine appropriate parameters by trial and error. To overcome these drawbacks, a learning rate maximization method (LeCun *et al.* 1993), learning rate adaptation rules (Jacobs 1988; Silva and Almeida 1990), smart algorithms such as QuickProp (Fahlman 1988) and RPROP (Riedmiller and Braun 1993), and second-order learning algorithms (Watrous 1987; Barnard 1992; Battiti 1992; Møller 1993b) based on nonlinear optimization techniques (Gill *et al.* 1981) have been proposed. Each has achieved a certain degree of

success. Among these approaches, we believe that second-order learning algorithms should be investigated more because they theoretically have excellent convergence properties (Gill *et al.* 1981). At present, however, they have two problems.

One is scalability. Second-order algorithms based on Levenberg-Marquardt or quasi-Newton methods cannot suitably increase in scale for large problems. Levenberg-Marquardt algorithms generally require a large amount of computation until convergence, even for midscale problems that involve many hundreds of weights. They require $O(N^2 m)$ operations to calculate the descent direction during any one iteration; N denotes the number of weights and m the number of examples. Standard quasi-Newton algorithms (Watrous 1987; Barnard 1992) are rarely applied to large-scale problems that involve more than many thousands of weights because they require N^2 storage space to maintain an approximation to an inverse Hessian matrix. In order to cope with this problem, the OSS algorithm (Battiti 1989) adopted the memoryless update (Gill *et al.* 1981); however, OSS may not provide desirable performance because the descent direction is calculated on the basis of a fast but rough approximation.

The other problem is the burden of step-length computation. A line search to calculate an adequate step length is indispensable for second-order algorithms that are based on quasi-Newton or conjugate gradient methods. Since an inaccurate line search may provide undesirable performance, a reasonably accurate line search is needed. However, an exact line search based on existing methods generally requires a nonnegligible computational load, at least a few number of function and/or gradient evaluations. Since it is widely recognized that successful convergence of conjugate gradient methods relies heavily on the accuracy of a line search, it seems rather difficult to increase the speed of conjugate gradient algorithms. In the SCG algorithm (Møller 1993b), although the step length is estimated by using the model trust region approach (Gill *et al.* 1981), this method can be regarded as an efficient one-step line search method based on a one-sided difference equation. Fortunately, successful convergence of quasi-Newton algorithms is theoretically guaranteed even with an inaccurate line search if certain conditions are satisfied (Powell 1976), but efficient convergence cannot always be expected. Thus, we believe that if the step length can be calculated with greater efficiency and reasonable accuracy, the quasi-Newton algorithms will work better from two aspects: processing efficiency and convergence.

For large-scale problems that include a large number of redundant training examples, on-line algorithms (LeCun *et al.* 1993), which perform a weight update for each single example, will work with greater efficiency than an off-line counterpart. Although second-order algorithms are basically not suited to perform on-line updating, waiting for a sweep of many examples before updating is not reasonable. Toward the improvement, several attempts have been made to introduce "pseudo-on-line" updating into second-order algorithms, where updates are performed on smaller subsets of data (Møller

1993c; Kuhn and Herzberg 1990). Thus, if a better second-order algorithm is developed, its pseudo-on-line version will work more efficiently.

This paper is organized as follows. Section 2 describes a new second-order learning algorithm based on a quasi-Newton method, called BPQ, where the descent direction is calculated on the basis of a partial BFGS update and a reasonably accurate step length is efficiently calculated as the minimal point of a second-order approximation. Section 3 evaluates BPQ's performance in comparison with other major learning algorithms.

2 BPQ Algorithm

2.1 The Problem. Let $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ be a set of examples, where $\mathbf{x}_t$ denotes an N-dimensional input vector and y_t a target value corresponding to $\mathbf{x}_t$. In a three-layer neural network, let h be the number of hidden units, $\mathbf{w}_i$ $(i = 1, \ldots, h)$ the weight vector between all the input units and the hidden unit i, and $\mathbf{w}_0 = (w_{00}, \ldots, w_{0h})^T$ the weight vector between all the hidden units and the output unit; w_{i0} means a bias term and x_{t0} is set to 1. Note that $\mathbf{a}^T$ denotes the transposed vector of $\mathbf{a}$. Hereafter, a vector consisting of all parameters, $(\mathbf{w}_0^T, \ldots, \mathbf{w}_h^T)^T$, is simply expressed as $\mathbf{\Phi}$; let $N(= nh + 2h + 1)$ be the dimension of $\mathbf{\Phi}$. Then, learning in the three-layer neural network can be defined as the problem of minimizing the following objective function:

$$f(\mathbf{\Phi}) = \frac{1}{2}\sum_{t=1}^{m}(y_t - z_t)^2, \tag{2.1}$$

where $z_t = z(\mathbf{x}_t; \mathbf{\Phi}) = w_{00} + \sum_{i=1}^{h} w_{0i}\sigma(\mathbf{w}_i^T\mathbf{x}_t)$. $\sigma(u)$ represents a sigmoidal function, $\sigma(u) = 1/(1 + e^{-u})$. Note that we do not employ nonlinear transformation at the output unit because it is not essential from the viewpoint of function approximation.

2.2 Quasi-Newton Method. A second-order Taylor expansion of $f(\mathbf{\Phi} + \Delta\mathbf{\Phi})$ about $\Delta\mathbf{\Phi}$ is given as $f(\mathbf{\Phi}) + (\nabla f(\mathbf{\Phi}))^T\Delta\mathbf{\Phi} + \frac{1}{2}(\Delta\mathbf{\Phi})^T\nabla^2 f(\mathbf{\Phi})\Delta\mathbf{\Phi}$. If $\nabla^2 f(\mathbf{\Phi})$ is positive definite, the minimal point of this expansion is given by $\Delta\mathbf{\Phi} = -(\nabla^2 f(\mathbf{\Phi}))^{-1}\nabla f(\Phi)$. Newton techniques minimize the objective function $f(\mathbf{\Phi})$ by iteratively calculating the descent direction, $\Delta\mathbf{\Phi}$ (Gill *et al.* 1981). However, since $O(N^3)$ operations are required to calculate $(\nabla^2 f(\mathbf{\Phi}))^{-1}$ directly, we cannot expect these techniques to increase suitably in scale for large problems. Quasi-Newton techniques, on the other hand, calculate a matrix $\mathbf{H}$ through iterations in order to approximate $(\nabla^2 f(\mathbf{\Phi}))^{-1}$. The basic algorithm is described as follows (Gill *et al.* 1981):

Step 1: Initialize $\mathbf{\Phi}_1$, set $\mathbf{H}_1 = \mathbf{I}$ ($\mathbf{I}$: identity matrix), and set $k = 1$.

Step 2: Calculate the current descent direction: $\Delta\mathbf{\Phi}_k = -\mathbf{H}_k\mathbf{g}_k$, where $\mathbf{g}_k = \nabla f(\mathbf{\Phi}_k)$.

Step 3: Terminate the iteration if a stopping criterion is satisfied.
Step 4: Calculate the step length λ_k that minimizes $f(\mathbf{\Phi}_k + \lambda \Delta \mathbf{\Phi}_k)$.
Step 5: Update the weights: $\mathbf{\Phi}_{k+1} = \mathbf{\Phi}_k + \lambda_k \Delta \mathbf{\Phi}_k$.
Step 6: If $k \equiv 0 \pmod{N}$, set $\mathbf{H}_{k+1} = \mathbf{I}$; otherwise, update $\mathbf{H}_{k+1}$.
Step 7: Set $k = k + 1$, return to Step 2.

2.3 Existing Methods for Calculating Descent Directions. Several methods for updating $\mathbf{H}_{k+1}$ have been proposed. Among them, the Broydon-Fletcher-Goldfarb-Shanno (BFGS) update (Fletcher 1980) was the most successful update in a number of studies. By putting $\mathbf{p}_k = \lambda_k \Delta \mathbf{\Phi}_k$ and $\mathbf{q}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$, the BFGS formula is given as

$$\mathbf{H}_{k+1} = \mathbf{H}_k - \frac{\mathbf{p}_k \mathbf{q}_k^T \mathbf{H}_k + \mathbf{H}_k \mathbf{q}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k} + \left(1 + \frac{\mathbf{q}_k^T \mathbf{H}_k \mathbf{q}_k}{\mathbf{p}_k^T \mathbf{q}_k}\right) \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k} . \tag{2.2}$$

In large-scale problems that involve more than many thousands of weights, maintaining the approximation matrix $\mathbf{H}$ becomes impractical because it requires N^2 storage space. In order to cope with this problem, the OSS (one-step secant) algorithm (Battiti 1989) adopted the memoryless BFGS update (Gill *et al.* 1981). By always taking the previous matrix $\mathbf{H}_k$ as the identity matrix ($\mathbf{H}_k = \mathbf{I}$), the descent direction of Step 2 is calculated as

$$\Delta \mathbf{\Phi}_{k+1} = -\mathbf{g}_{k+1} + \frac{\mathbf{p}_k \mathbf{q}_k^T \mathbf{g}_{k+1} + \mathbf{q}_k \mathbf{p}_k^T \mathbf{g}_{k+1}}{\mathbf{p}_k^T \mathbf{q}_k} - \left(1 + \frac{\mathbf{q}_k^T \mathbf{q}_k}{\mathbf{p}_k^T \mathbf{q}_k}\right) \frac{\mathbf{p}_k \mathbf{p}_k^T \mathbf{g}_{k+1}}{\mathbf{p}_k^T \mathbf{q}_k} . \tag{2.3}$$

Clearly equation 2.3 can be calculated with $O(N)$ multiplications and storage space. However, OSS may not provide desirable performance because the descent direction is calculated on the basis of the above substitution. In our early experiments, this method worked rather poorly in comparison with the partial BFGS update proposed in the next section.

2.4 New Method for Calculating Descent Directions. In this section, we propose a partial BFGS update with $2Ns$ memory ($s \ll N$), where the search directions are exactly equivalent to those of the original BFGS update during the first $s+1$ iterations. The partiality parameter s means the length of the history described below. By putting $\mathbf{r}_k = \mathbf{H}_k \mathbf{q}_k$, we have

$$\mathbf{H}_k \mathbf{g}_{k+1} = \mathbf{H}_k \mathbf{q}_k + \mathbf{H}_k \mathbf{g}_k \;=\; \mathbf{r}_k - \frac{\mathbf{p}_k}{\lambda_k} .$$

Then, the descent direction based on equation 2.2 can be calculated as

$$\Delta \mathbf{\Phi}_{k+1} = -\mathbf{H}_{k+1} \mathbf{g}_{k+1}$$

$$= -\mathbf{r}_k + \frac{\mathbf{p}_k}{\lambda_k} + \frac{\mathbf{p}_k \mathbf{r}_k^T \mathbf{g}_{k+1} + \mathbf{r}_k \mathbf{p}_k^T \mathbf{g}_{k+1}}{\mathbf{p}_k^T \mathbf{q}_k}$$

$$- \left(1 + \frac{\mathbf{q}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{q}_k}\right) \frac{\mathbf{p}_k \mathbf{p}_k^T \mathbf{g}_{k+1}}{\mathbf{p}_k^T \mathbf{q}_k} . \tag{2.4}$$

After calculating $\mathbf{r}_k$, equation 2.4 can be calculated using $O(N)$ multiplications and storage space, just like the memoryless BFGS update. Now, we show that it is possible to calculate $\mathbf{r}_k$ within $O(Ns)$ multiplications and $2Ns$ storage space by using the partial BFGS update.

Here, we assume $k \leq s$. When $k = 1$, $\mathbf{r}_1$ $(= \mathbf{H}_1 \mathbf{q}_1 = \mathbf{g}_2 - \mathbf{g}_1)$ can be calculated only by subtractions. When $k > 1$, we assume that each of $\mathbf{r}_1, \ldots, \mathbf{r}_{k-1}$ has been calculated and stored. Note that for $i < k$,

$$\alpha_i = \frac{1}{\mathbf{p}_i^T \mathbf{q}_i} \text{ and } \beta_i = \alpha_i (1 + \alpha_i \mathbf{q}_i^T \mathbf{r}_i)$$

have already been calculated during the iterations. Thus, by recursively applying equation 2.2, $\mathbf{r}_k$ can be calculated with $O(Nk)$ multiplications and $2Nk$ storage space, as follows:

$$\begin{aligned} \mathbf{r}_k &= \mathbf{H}_k \mathbf{q}_k \\ &= \mathbf{H}_{k-1} \mathbf{q}_k - \alpha_{k-1} \mathbf{p}_{k-1} \mathbf{r}_{k-1}^T \mathbf{q}_k - \alpha_{k-1} \mathbf{r}_{k-1} \mathbf{p}_{k-1}^T \mathbf{q}_k + \beta_{k-1} \mathbf{p}_{k-1} \mathbf{p}_{k-1}^T \mathbf{q}_k \\ &= \mathbf{q}_k + \sum_{i=1}^{k-1} (-\alpha_i \mathbf{p}_i \mathbf{r}_i^T \mathbf{q}_k - \alpha_i \mathbf{r}_i \mathbf{p}_i^T \mathbf{q}_k + \beta_i \mathbf{p}_i \mathbf{p}_i^T \mathbf{q}_k) . \end{aligned} \tag{2.5}$$

Next, when the number of iterations exceeds $s+1$, we have two alternatives: restarting the update by discarding the accumulated vectors or continuing the update by using the latest vectors. In both cases, equation 2.5 can be calculated within $O(Ns)$ multiplications and $2Ns$ storage space. Therefore, it has been shown that equation 2.4 can be calculated within $O(Ns)$ multiplications and $2Ns$ storage space. In our experiments, the former update was employed because the latter worked poorly when s was small.

Although the idea of partial quasi-Newton methods has been briefly described (Luenberger 1984), strictly speaking, our update is different. The earlier proposal intended to store the vectors $\mathbf{p}_i$ and $\mathbf{q}_i$, but our update stores the vectors $\mathbf{p}_i$ and $\mathbf{r}_i$. The immediate advantage of this partial update over the original BFGS update is the applicability to large-scale problems. Even if N is very large, by setting s to an adequate small value with respect to the amount of available storage space, our update will work. On the other hand, the probable advantage over the memoryless BFGS update is the superiority of the convergence property. However, this claim should be examined through a wide range of experiments. Note that when $s = 0$, the partial BFGS update always gives the gradient direction; when $s = 1$, it corresponds to the memoryless BFGS update.

2.5 Existing Methods for Calculating Step Lengths. In Step 4, since λ is the only variable in f, we can express $f(\mathbf{\Phi}+\lambda\Delta\mathbf{\Phi})$ simply as $\zeta(\lambda)$. Calculating λ, which minimizes $\zeta(\lambda)$, is called a line search. Among a number of possible line search methods, we considered the following typical three methods: a fast but inaccurate one, a moderately accurate one, and an exact but slow one.

The first method is explained below. By using quadratic interpolation (Gill *et al.* 1981; Battiti 1989), we can derive a fast line search method that only guarantees $\zeta(\lambda) < \zeta(0)$. Let λ_1 be an initial value for a step length. If $\zeta(\lambda_1) < \zeta(0)$, λ_1 becomes the resultant step length; otherwise, by considering a quadratic function $h(\lambda)$ that satisfies the conditions $h(0) = \zeta(0)$, $h(\lambda_1) = \zeta(\lambda_1)$, and $h'(0) = \zeta'(0)$, we get the following approximation of $\zeta(\lambda)$:

$$\zeta(\lambda) \approx h(\lambda) \;=\; \zeta(0) + \zeta'(0)\lambda + \frac{\zeta(\lambda_1) - \zeta(0) - \zeta'(0)\lambda_1}{\lambda_1^2}\lambda^2.$$

Since $\zeta(\lambda_1) \geq \zeta(0)$ and $\zeta'(0) < 0$, the minimal point of $h(\lambda)$ is given by

$$\lambda_2 = -\frac{\zeta'(0)\lambda_1^2}{2(\zeta(\lambda_1) - \zeta(0) - \zeta'(0)\lambda_1)}\ . \tag{2.6}$$

Note that $0 < \lambda_2 < \lambda_1$ is guaranteed by equation 2.6. Thus, by iterating this process until $\zeta(\lambda_\nu) < \zeta(0)$, we can always find a λ_ν that satisfies $\zeta(\lambda_\nu) < \zeta(0)$, where ν denotes the number of iterations. Here, the initial value of λ_1 is set to 1 because the optimal step length is near 1 when $\mathbf{H}$ closely approximates $(\nabla^2 f(\mathbf{\Phi}))^{-1}$. Hereafter, a quasi-Newton algorithm based on the original or partial BFGS update in combination with this fast but inaccurate line search is called BFGS1.

By using quadratic extrapolation (Fletcher 1980), we can derive a moderately accurate line-search method that guarantees $\zeta'(\lambda_\nu) > \gamma_1\zeta'(\lambda_{\nu-1})$ as well as $\zeta(\lambda_\nu) < \zeta(\lambda_{\nu-1})$, where γ_1 is a small constant (e.g., 0.1). Namely, when λ_ν does not satisfy the stopping criterion, if $\zeta(\lambda_\nu) \geq \zeta(\lambda_{\nu-1})$, $\lambda_{\nu+1}$ is calculated using equation 2.6; otherwise, by considering an extrapolation to the slopes $\zeta'(\lambda_{\nu-1})$ and $\zeta'(\lambda_\nu)$, the appropriate expression for the minimizing value $\lambda_{\nu+1}$ is

$$\lambda_{\nu+1} = \lambda_\nu - \zeta'(\lambda_\nu)\frac{\lambda_\nu - \lambda_{\nu-1}}{\zeta'(\lambda_\nu) - \zeta'(\lambda_{\nu-1})}\ . \tag{2.7}$$

However, if $\zeta'(\lambda_\nu) \leq \zeta'(\lambda_{\nu-1})$, then $\lambda_{\nu+1}$ does not exist; thus, $\lambda_{\nu+1}$ is set to $\lambda_{\nu-1}+\gamma_2(\lambda_\nu-\lambda_{\nu-1})$, where γ_2 is an adequate value (e.g., 9). In this method, λ_0 is set to 0, and λ_1 is estimated as $\min(1, -2\zeta'(0)^{-1}(f(\mathbf{\Phi}_{current}) - f(\mathbf{\Phi}_{previous})))$. Note that to use this estimate of λ_1 for the first method is not good strategy because it does not have an extrapolation process, but λ_1 can be a very small value. Hereafter, a quasi-Newton algorithm based on the original or partial

BFGS update in combination with this moderately accurate line search is called BFGS2.

An exact but slow line search method can be constructed by iteratively using the BFGS2 line search method until a stopping criterion is met, for example, $\|\zeta'(\lambda_v)\| < 10^{-8}$. The difference from the BFGS2 method is that equation 2.7 is used for interpolation as well as extrapolation. Hereafter, a quasi-Newton algorithm based on the original or partial BFGS update in combination with this exact but slow line search is called BFGS3.

2.6 New Method for Calculating Step Lengths. Here we propose a new method for calculating a reasonably accurate step-length λ in Step 4.

2.6.1 Basic Procedure. A second-order Taylor approximation of $\zeta(\lambda)$ is given as

$$\zeta(\lambda) \approx \zeta(0) + \zeta'(0)\lambda + \frac{1}{2}\zeta''(0)\lambda^2.$$

When $\zeta'(0) < 0$ and $\zeta''(0) > 0$, the minimal point of this approximation is given by

$$\lambda = -\frac{\zeta'(0)}{\zeta''(0)} \left(= -\frac{(\nabla f(\mathbf{\Phi}))^T \Delta\mathbf{\Phi}}{(\Delta\mathbf{\Phi})^T \nabla^2 f(\mathbf{\Phi}) \Delta\mathbf{\Phi}} \right). \tag{2.8}$$

Other cases will be considered in the next section.

For the three-layer neural networks defined by equation 2.1, we can efficiently calculate $\zeta'(0)$ and $\zeta''(0)$ as follows. By differentiating $\zeta(\lambda)$ and substituting 0 for λ, we obtain

$$\zeta'(0) = -\sum_{t=1}^{m}(y_t - z_t)z'_t \text{ and } \zeta''(0) = \sum_{t=1}^{m}((z'_t)^2 - (y_t - z_t)z''_t).$$

Now that the derivative of $z_t = z(\mathbf{x}_t; \mathbf{\Phi})$ is defined as $\frac{d}{d\lambda} z(\mathbf{x}_t; \mathbf{\Phi} + \lambda\Delta\mathbf{\Phi})_{|\lambda=0}$, we obtain

$$z'_t = \Delta w_{00} + \sum_{i=1}^{h}(\Delta w_{0i}\sigma_{it} + w_{0i}\sigma'_{it}) \text{ and } z''_t = \sum_{i=1}^{h}(2\Delta w_{0i}\sigma'_{it} + w_{0i}\sigma''_{it}),$$

where $\sigma_{it} = \sigma(\mathbf{w}_i^T\mathbf{x}_t)$, $\sigma'_{it} = \sigma_{it}(1 - \sigma_{it})(\Delta\mathbf{w}_i)^T\mathbf{x}_t$, $\sigma''_{it} = \sigma'_{it}(1 - 2\sigma_{it})(\Delta\mathbf{w}_i)^T\mathbf{x}_t$, and Δw_{ij} denotes the change of w_{ij}, calculated in Step 2.

Now we consider the computational complexity of calculating the step length using equation 2.8. Clearly, $(\Delta\mathbf{w}_i)^T\mathbf{x}_t$ must be calculated for each pair of hidden unit i and input $\mathbf{x}_t$; thus, since the number of hidden units is h and the number of inputs is m, at least nhm multiplications are required. Since the order of multiplications required to calculate the remainder is $O(hm)$, and $N = nh + 2h + 1$, the total complexity of the calculation is $Nm + O(hm)$.

This algorithm can be generalized to multilayered networks with other differentiable activation functions; thus, it is applicable to recurrent networks (Rumelhart *et al.* 1986). Consider a hidden unit τ connected to the output layer. We assume that its output value is defined by $v = a(\mathbf{w}^T\mathbf{u})$, where $\mathbf{u}$ is the output values of the units connected to the unit τ, $\mathbf{w}$ equals the weights attached to these connections, and $a(\cdot)$ is an activation function. Then the first- and second-order derivatives of v with respect to λ are calculated as

$$v' = a'(\mathbf{w}^T\mathbf{u})(\Delta\mathbf{w}^T\mathbf{u} + \mathbf{w}^T\mathbf{u}'),$$
$$v'' = a''(\mathbf{w}^T\mathbf{u})(\Delta\mathbf{w}^T\mathbf{u} + \mathbf{w}^T\mathbf{u}')^2 + a'(\mathbf{w}^T\mathbf{u})(2\Delta\mathbf{w}^T\mathbf{u}' + \mathbf{w}^T\mathbf{u}'').$$

Thus, by successively applying these formulas in reverse, we can calculate the step length for multilayered networks. Note that if u_i is an output value of an input unit, then $u_i' = u_i'' = 0$.

2.6.2 Coping with Undesirable Cases. In the above, we assumed $\zeta'(0) < 0$. When $\zeta'(0) > 0$, the value of the objective function cannot be reduced along the search direction; thus, we set $\Delta\mathbf{\Phi}_k$ to $-\nabla f(\mathbf{\Phi}_k)$ and restart the update by discarding the accumulated vectors $(\mathbf{p}, \mathbf{r})$. Note that $\zeta'(0) < 0$ is guaranteed by such a setting unless $\|\nabla f(\mathbf{\Phi}_k)\| = 0$ because $\zeta'(0) = (\nabla f(\mathbf{\Phi}_k))^T\Delta\mathbf{\Phi}_k = -\|\nabla f(\mathbf{\Phi}_k)\|^2 < 0$.

When $\zeta'(0) < 0$ and $\zeta''(0) \leq 0$, equation 2.8 gives a negative value or infinity. To avoid this situation, we employ the Gauss-Newton technique. The first-order approximation of $z_t = z(\mathbf{x}_t; \mathbf{\Phi} + \lambda\Delta\mathbf{\Phi})$ is $z_t + z_t'\lambda$. Then, $\zeta(\lambda)$ of the next iteration can be approximated by

$$\zeta(\lambda) \approx \frac{1}{2}\sum_{t=1}^{m}(y_t - (z_t + z_t'\lambda))^2 \;=\; \zeta(0) + \zeta'(0)\lambda + \frac{1}{2}\sum_{t=1}^{m}(z_t')^2\lambda^2.$$

The minimal point of this approximation is given by

$$\lambda = -\frac{\zeta'(0)}{\sum_{t=1}^{m}(z_t')^2}\,. \tag{2.9}$$

Clearly, equation 2.9 always gives a positive value when $\zeta'(0) < 0$.

In many cases, it is useful from a practical sense to limit the maximum change in $\mathbf{\Phi}$, which should be done during any one iteration (Gill *et al.* 1981). Here, if $\|\lambda\Delta\mathbf{\Phi}\| > 1.0$, λ is set to $\|\Delta\mathbf{\Phi}\|^{-1}$.

Since λ is calculated on the basis of the approximation, we cannot always reduce the value of the objective function, $\zeta(\lambda)$. When $\zeta(\lambda) \geq \zeta(0)$, we employ the fast line search given by equation 2.6.

2.6.3 Summary of Step-Length Calculation. By integrating the above procedures, we can specify Step 4 as follows.

Step 4.1: If $\zeta'(0) > 0$, set $\Delta\mathbf{\Phi}_k = -\nabla f(\mathbf{\Phi}_k)$ and $k = 1$.

Step 4.2: If $\zeta''(0) > 0$, calculate λ using equation 2.8; otherwise, calculate λ using equation 2.9.

Step 4.3: If $\|\lambda\Delta\mathbf{\Phi}_k\| > 1.0$, set $\lambda = \|\Delta\Phi_k\|^{-1}$.

Step 4.4: If $\zeta(\lambda) > \zeta(0)$, calculate λ using equation 2.6 until $\zeta(\lambda) < \zeta(0)$.

Hereafter, the quasi-Newton algorithm based on our partial BFGS update in combination with our step-length calculation is called BPQ (Bp based on partial Quasi-Newton). Incidentally, a modified OSS algorithm, OSS2 (a combination of the memoryless BFGS update and our step-length calculation), may be another good algorithm and will be evaluated in the experiments.

2.7 Computational Complexity. We consider the computational complexity of BPQ and other algorithms with respect to one-iteration in which every training example is presented once. In off-line BP, the complexity (number of multiplications) to calculate the objective function is $nhm + O(hm)$ and the complexity for the gradient vector is $nhm + O(hm)$. Thus, since $N = nh + 2h + 1$, the complexity for off-line BP is $2Nm + O(hm)$. Here, the complexity for a weight update is just N (or $2N$ if momentum term is used), and is safely negligible.

In this article, we define one-iteration of on-line BP as m updates sweeping all examples once. In addition to the above complexity, since on-line BP performs a weight update for each single example, the learning rate is multiplied to each element of the gradient vectors Nm times in one-iteration; thus, the complexity for on-line BP is $3Nm + O(hm)$. In the case of on-line BP with momentum term, the momentum factor is also multiplied to each element of the previous modification vectors Nm times in one-iteration; thus, the complexity for on-line momentum BP is $4Nm + O(hm)$.

In addition to the complexity for off-line BP, BPQ calculates the descent direction based on the partial BFGS update with a history of at most s iterations and also calculates the step length. The complexity of the former calculation is $O(Ns)$ (see Section 2.4) and that of the latter is $Nm + O(hm)$ (see Section 2.6.1). Here, note that the computational complexity to calculate the objective function can be reduced from $Nm + O(hm)$ to $O(hm)$ for three-layer networks. This is because in the next iteration, the output value of each hidden unit is given by $\sigma(\mathbf{w}_i^T\mathbf{x}_t + \lambda(\Delta\mathbf{w}_i)^T\mathbf{x}_t)$, but $(\Delta\mathbf{w}_i)^T\mathbf{x}_t$ is already calculated when the step length is calculated. Thus, the total complexity for BPQ is $2Nm + O(Ns) + O(hm)$. To reduce the generalization error for an unseen example, m should be larger than N. Since s is smaller than N, the complexity of $O(Ns)$ usually becomes much smaller than that of $2Nm$, and the complexity for BPQ remains almost equivalent to that of off-line BP.

A general method for calculating the denominator of equation 2.8 has been proposed (Pearlmutter 1994; Møller 1993a); after calculating $\nabla^2 f(\mathbf{\Phi})\Delta\mathbf{\Phi}$,

the denominator is calculated by using an inner product. The result is mathematically the same as our step-length calculation, but the computational complexity of this method is much larger than that of our method, at least in the case of three-layer networks as shown below. By using Pearlmutter's operator, which is defined by $\Re_{\Delta\mathbf{\Phi}}\{f(\mathbf{\Phi})\} = \frac{\partial}{\partial\lambda} f(\Phi+\lambda\Delta\mathbf{\Phi})_{|\lambda=0}$ (Pearlmutter 1994), we can see that $\Re_{\Delta\mathbf{\Phi}}\{\frac{\partial}{\partial w_{ij}} f(\mathbf{\Phi})\}$ is an element of $\nabla^2 f(\mathbf{\Phi})\Delta\mathbf{\Phi}$ because $\Re_{\Delta\mathbf{\Phi}}\{\nabla f(\mathbf{\Phi})\} = \nabla^2 f(\mathbf{\Phi})\Delta\mathbf{\Phi}$. Now, considering the case of a weight between the output unit and the hidden unit i, which is expressed as w_{0i}, we obtain

$$\Re_{\Delta\Phi}\left\{\frac{\partial}{\partial w_{0i}} f(\mathbf{\Phi})\right\} = \Re_{\Delta\mathbf{\Phi}}\left\{-\sum_{t=1}^{m}(y_t - z_t)\sigma_{it}\right\}$$

$$= \sum_{t=1}^{m}(\Re_{\Delta\mathbf{\Phi}}\{z_t\}\sigma_{it} - (y_t - z_t)\Re_{\Delta\Phi}\{\sigma_{it}\}),$$

where $\Re_{\Delta\mathbf{\Phi}}\{z_t\} = \Delta w_{00} + \sum_{i=1}^{h}(\Delta w_{0i}\sigma_{it} + w_{0i}\Re_{\Delta\mathbf{\Phi}}\{\sigma_{it}\})$ and $\Re_{\Delta\mathbf{\Phi}}\{\sigma_{it}\} = \sigma_{it}(1-\sigma_{it})(\Delta\mathbf{w}_i)^T\mathbf{x}_t$. Clearly, just like our method, $(\Delta\mathbf{w}_i)^T\mathbf{x}_t$ must be calculated for each pair of hidden unit i and input $\mathbf{x}_t$; thus, a minimum of nhm multiplications is required. Additionally, a complexity of at least $O(m)$ is required for calculating $\Re_{\Delta\Phi}\{\frac{\partial}{\partial w_{0i}} f(\mathbf{\Phi})\}$. Therefore, since a similar argument can be applied to other weights and the total number of the weights is N, the complexity of the existing method becomes $nhm + O(Nm)$. On the other hand, the complexity of our method is $nhm + O(hm)$, which is more efficient because $N = nh + 2h + 1$.

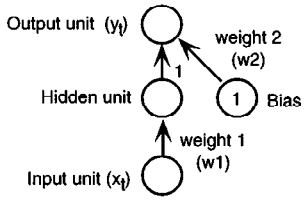
Although the SCG (scaled conjugate gradient) algorithm (Møller 1993b) estimates the step length by using the model trust region approach, this method can be regarded as an efficient one-step line search method based on a one-sided difference equation. The step length calculated from equation 2.8 is approximated by

$$\nabla^2 f(\mathbf{\Phi})\Delta\mathbf{\Phi} \approx \frac{\nabla f(\mathbf{\Phi} + \delta\Delta\mathbf{\Phi}) - \nabla f(\mathbf{\Phi})}{\delta},$$

where δ is defined as $\delta = \delta_0\|\Delta\mathbf{\Phi}\|^{-1}$ and δ_0 is a small constant. Clearly, the complexity of SCG is approximately double that of BP because the gradient vector $\nabla f(\mathbf{\Phi} + \delta\Delta\mathbf{\Phi})$ must be calculated during one-iteration. If the difference parameter δ is approximately equal to the optimal step-length λ, SCG will provide a more faithful picture of the error surface than our method; however, since our purpose is to estimate the optimal step-length λ, we generally do not know such δ.

3 Experiments

This section evaluates BPQ's performance in comparison with many other learning algorithms through experiments that use three types of problems.



Input value (x_t)	Target value (y_t)
$x_1 = 1$	$y_1 = 0.73$
$x_2 = 2$	$y_2 = 0.88$
$x_3 = 3$	$y_3 = 0.95$
$x_4 = 4$	$y_4 = 0.98$
$x_5 = 5$	$y_5 = 0.99$

Figure 1: Two-weight problem.

3.1 Two-Weight Problem. In order to evaluate learning efficiency graphically, we designed a simple learning problem where only two weights were adjustable. Figure 1 describes the problem. In a three-layer network, each layer consists of only one unit, and the weight between the hidden unit and the output unit is fixed at 1. In this problem, the weight between the input and hidden units is expressed as w_1, the weight corresponding to the bias term at the output unit is expressed as w_2, and an activation function of a hidden unit is assumed to be $\sigma(x) = 1/(1 + \exp(-x))$. Each target value y_t shown in Figure 1 was calculated from the corresponding input value x_t by setting $(w_1, w_2) = (1, 0)$. Thus, the global minimum point was given by these weight values.

In this experiment, BPQ was compared with eight other learning algorithms: on-line BP with momentum term, off-line BP, off-line BP with momentum term, BP with learning rate adaptation (Silva and Almeida 1990), SCG (Møller 1993b), BFGS1, BFGS2, and BFGS3. Note that the memoryless BFGS update is equivalent to the original BFGS update when $N = 2$. The parameters of each algorithm were determined by trial and error or set to the values recommended by the inventors (Silva and Almeida 1990; Møller 1993b). For on-line momentum BP, the learning rate and the momentum factor were set to $\eta = 0.05$ and $\alpha = 0.9$. For off-line BP, the learning rate η was set to 0.1. For off-line momentum BP, the learning rate and the momentum factor were set to $\eta = 0.025$ and $\alpha = 0.9$. For adaptive BP, the increase and the decrease factors were set to $u = 1.1$ and $d = u^{-1}$, and the initial update value η_0 was set to 0.01. For SCG, the constant δ_0 was set to 10^{-4}, and the initial scaling parameter λ_1 was set to 10^{-6}.

Figures 2 and 3 show the learning trajectories on the error surface with respect to w_1 and w_2 during 100 iterations starting at $(w_1, w_2) = (-0.5, 0)$, where MSE stands for mean squared error:

$$\mathrm{MSE} = \frac{1}{5}\sum_{t=1}^{5}(y_t - (\sigma(w_1 x_t) + w_2))^2.$$

(These experiments were done on a Power Macintosh 8100 / 100 computer.)

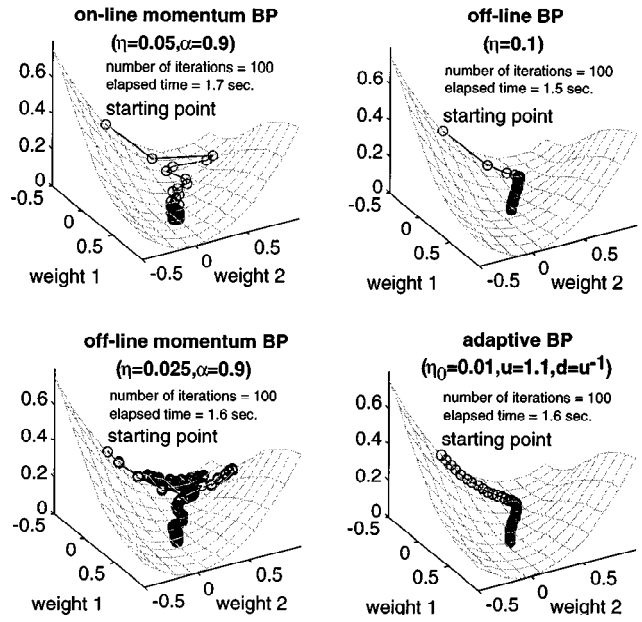


Figure 2: Learning trajectories of first-order algorithms.

Figure 2 shows that the search of these first-order algorithms went very slowly along the curved valley floor. Up to 100 iterations, any first-order algorithm was unable to reach the minimum point. This may be due to the inherent difficulty of first-order methods; since the directions of the successive gradient vectors are almost opposite near the valley floor, even learning rate adaptation by sign changes of the last two gradients cannot improve the learning efficiency. Note that the trajectories of BPs oscillate widely when a larger η is used.

On the other hand, Figure 3 shows that SCG, BFGS1, BFGS2, BFGS3, and BPQ found the minimum point within 100 iterations. In comparison with BPQ, SCG required a larger number of iterations before reaching the minimum point. BFGS1 required the largest number of iterations, showing that the accuracy of the step-length calculation is important for solving this problem. BFGS2 worked very well, almost identical to BPQ. The number of iterations for BFGS3 was slightly smaller than that for BPQ, but BFGS3 required much more computation time due to an exact line search. These

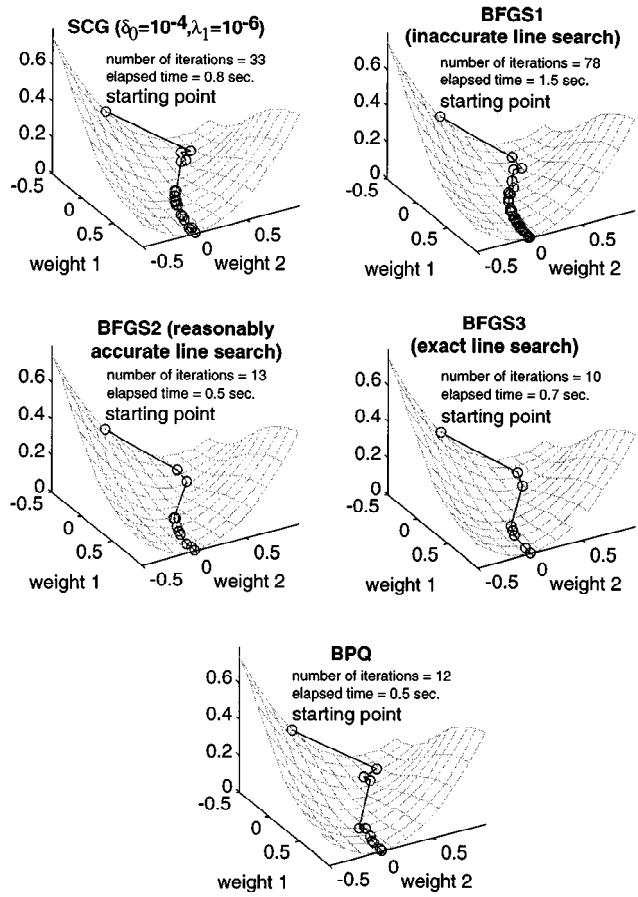


Figure 3: Learning trajectories of second-order algorithms.

experimental results indicate that second-order methods will work well even if an error surface forms a curved valley floor.

3.2 Parity Problem. Since parity problems have been widely used as benchmarks, the eight-bit parity problem was used to compare BPQ with seven other algorithms: on-line momentum BP, adaptive BP, SCG, OSS2, BFGS1, BFGS2, and BFGS3. Since the scale of the problem was quite small, BPQ employed the original BFGS update; thus $s = N$. In the experiments, we used a three-layer network with eight hidden units ($h = 8$, $N = 82$).

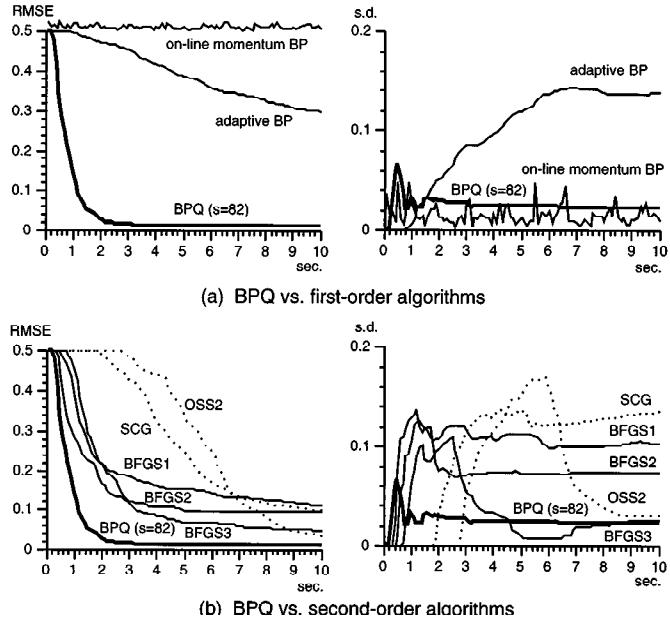


Figure 4: Convergence for the 8-bit parity problem.

All possible input patterns were used as training examples ($m = 256$), and the target values were set to 1 or 0. The parameters of each algorithm were exactly the same as in the previous experiment, except that the learning rate η was set to 0.1 for on-line momentum BP. In all the algorithms, the initial values for the weights were randomly generated in the range of $[-1, 1]$. In the experiments, the maximum CPU processing time was set to 100 seconds. The iteration was terminated when $\|\nabla f(\mathbf{w})\| < 10^{-8}$. (These experiments were done on an HP9000/735 computer.)

Figure 4a shows convergence property of BPQ and first-order algorithms with respect to the average RMSE (root mean squared error): $\sqrt{2f(\mathbf{w})/m}$) and the standard deviation over 100 trials for each algorithm. Among these algorithms, BPQ converged the quickest, allowing us to stop the iteration at an early stage. Figure 4b compares convergence property of BPQ with second-order algorithms. In comparison with BPQ, the convergence of the other second-order algorithms was slow. This shows that the step-length calculation plays an important role for quasi-Newton methods.

3.3 Practical Problem. In order to evaluate how BPQ works on large-scale problems, we performed experiments on a speech synthesis problem by using a data set used for parrot-like speaking (Nakano *et al.* 1995). This learning problem is a sort of nonlinear autoregression; the input vector consists of a feature vector (10 values) and an acoustic wave (8 values), and the target output value is the next value of the acoustic wave; the total number of input units is 18 ($n = 18$). In these experiments, the number of hidden units was set to 36 ($h = 36$); thus, the number of weights was 721 ($N = 721$). By using 12,800 training examples ($m = 12,800$), BPQ was compared with eight other algorithms: on-line BP, on-line momentum BP, adaptive BP, SCG, OSS2, BFGS1, BFGS2, and BFGS3. In the algorithms, the initial values for the weights between the input and hidden units were randomly generated in the range of $[-1, 1]$. The values for the weights between the hidden and output units were set to 0, but the bias value at the output unit was set to the average output value of all training examples. In the experiments, the maximum number of iterations was set to 100, and the average RMSE was used to evaluate the convergence property. (These experiments were also done on an HP9000/735 computer.)

Figure 5a compares BPQ with the first-order algorithms using the convergence property with respect to the average RMSE and standard deviation over 10 trials for each algorithm. Note that standard deviation for adaptive BP is not shown because it was always 0. For on-line BP, the learning rate η was set to 1.0, the best value in our experiments; however, the learning curve oscillated widely. For on-line momentum BP whose momentum factor α was fixed at 0.9, the learning rate η was set to 0.1, the best value in our experiments; although the learning curve did not oscillate widely, the convergence property was very similar to that of on-line BP. For adaptive BP, the learning rate η was set to 0.1 or 1.0; both values gave similar and rather poor results. For BPQ, the parameter s was set to 50; convergence was the quickest, allowing us to stop the iteration at an early stage. Figure 5b compares BPQ with other second-order algorithms using the average values over 10 trials for each algorithm. In comparison to BPQ, BFGS1 worked rather poorly; BFGS3 required more processing time to achieve the level equal to the best RMSE of BPQ, while BFGS2 worked better than BFGS3, rather close to BPQ.

Figure 5c compares the average one-iteration processing time of all algorithms. One-iteration time of OSS2 or BPQ was almost equivalent to that of adaptive BP. On-line BP was about 1.3 times slower due to weight updates for each single example; on-line momentum BP was almost twice as slow due to additional multiplication of the momentum factor. BFGS1 required a half more processing since it sometimes required a few function evaluations in order to shorten the step lengths; BFGS2 was almost four times slower due to an additional line search calculation to meet its stopping criterion; BFGS3 was almost seven times slower due to an exact line search calculation. SCG was almost twice as slow due to an additional gradient calculation.

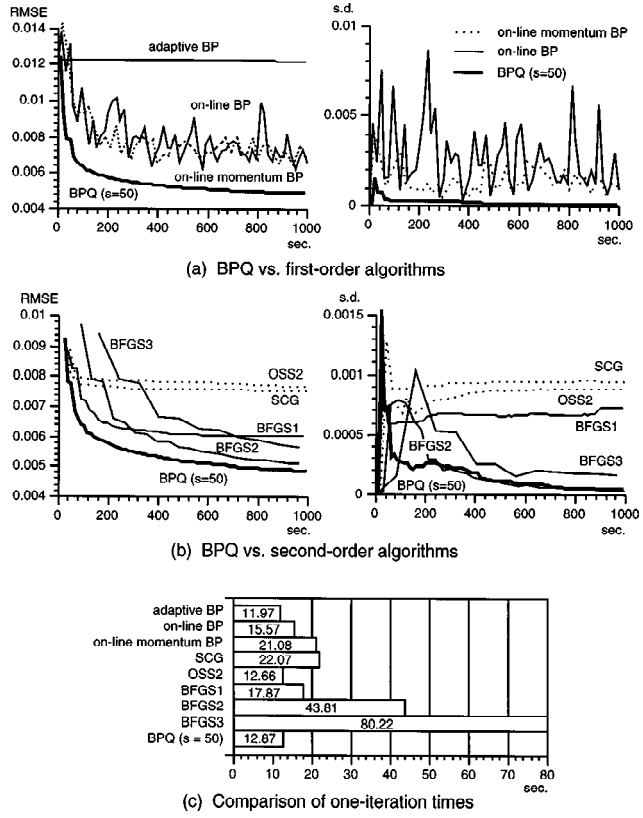


Figure 5: Convergence for a practical problem.

Figure 6 shows the influence of the partiality parameter s on the convergence property for each of the four line search algorithms. Each curve is drawn using the average RMSE over 10 trials. In general, the inference was small compared with the difference of line search methods. BPQ with $s = 5$ worked as nicely as with $s = 50$, while BPQ with $s = 2$ worked poorly compared with BPQ with $s \geq 5$. Consequently, it was shown that an efficient and accurate step-length calculation plays an important role for the convergence of quasi-Newton algorithms, while the partial BFGS update with a proper s greatly saves storage space without losing the convergence performance.

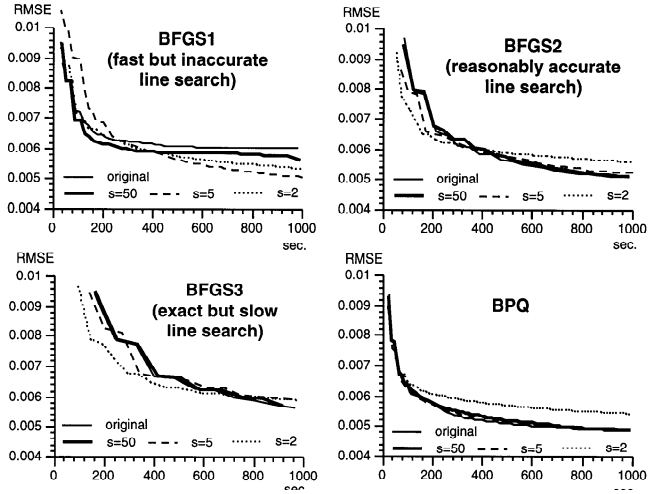


Figure 6: Effect of partiality parameter *s*.

4 Conclusion

We have proposed a small-memory efficient second-order learning algorithm called BPQ for three-layer neural networks. In BPQ, the search direction is calculated on the basis of a partial BFGS update, and a reasonably accurate step length is efficiently calculated as the minimal point of a second-order approximation. In our experiments that used a two-weight problem, a parity problem, and a speech synthesis problem, BPQ worked better than major learning algorithms. Moreover, it turned out that an efficient and accurate step-length calculation plays an important role for the convergence of quasi-Newton algorithms, while the partial BFGS update greatly saves storage space without losing the convergence performance. In the future, we plan to do further comparisons using a wider variety of problems, including large-scale classification problems.

Acknowledgments

We thank anonymous referees for many helpful suggestions and comments.

References

Barnard, E. 1992. Optimization for training neural nets. *IEEE Trans. Neural Networks* **3**(2), 232–240.

Battiti, R. 1989. Accelerating back-propagation learning: Two optimization methods. *Complex Systems* **3**(4), 331–342.

Battiti, R. 1992. First- and second-order methods for learning between steepest descent and Newton's method. *Neural Comp.* **4**(2), 141–166.

Fahlman, S. 1988. Faster-learning variations on back-propagation: an empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*, pp. 38–51. San Mateo, CA.

Fletcher, R. 1980. *Practical Methods of Optimization*. Vol. 1. John Wiley, New York.

Gill, P., Murray, W., and Wright, M. 1981. *Practical Optimization*. Academic Press, London.

Jacobs, R. 1988. Increased rates of convergence through learning rate adaptation. *Neural Networks* **1**(4), 295–307.

Kuhn, G., and Herzberg, P. 1990. Some variations on training recurrent neural networks. In *Proceedings of CAIP Neural Networks Workshop*, pp. 15–17, Rutgers University, New Brunswick, NJ.

LeCun, Y., Simard, P., and Pearlmutter, B. 1993. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In *Neural Information Processing Systems 5*, S. Hanson, J. Cowan, and C. Giles, eds., pp. 156–163. Morgan Kaufmann, San Mateo, CA.

Luenberger, D. 1984. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA.

Møller, M. 1993a. Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in O(N) time. Tech. Rep. DAIMI PB-432, Computer Science Department, Aarthus University.

Møller, M. 1993b. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks* **6**(4), 525–533.

Møller, M. 1993c. Supervised learning on large redundant training sets. *International J. Neural Systems* **4**(1), 15–25.

Nakano, R., Ueda, N., Saito, K., and Yamada, T. 1995. Parrot-like speaking using optimal vector quantization. In *Proc. IEEE International Conference on Neural Networks*, Parse, Australia.

Pearlmutter, B. 1994. Fast exact multiplication by the Hessian. *Neural Comp.* **6**(1), 147–160.

Powell, M. 1976. Some global convergence properties of a variable metric algorithm for minimization without exact line searches. In *Nonlinear Programing, SIAM-AMS Proceedings*, Vol. 9, Providence, RI.

Riedmiller, M., and Braun, H. 1993. A direct adaptive method for faster back-propagation learning: The rprop algorithm. In *Proc. IEEE International Conference on Neural Networks*, San Francisco.

Rumelhart, D., Hinton, G., and Williams, R. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing*, D. Rumelhart and J. McClelland, eds., pp. 318–362. MIT Press, Cambridge, MA.

Silva, F., and Almeida, L. 1990. Speeding up backpropagation. In *Advanced Neural Computers*, R. Eckmiller, ed., pp. 151–160. North-Holland, Amsterdam.

Watrous, R. 1987. Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. In *Proc. IEEE International Conference on Neural Networks*, pp. II619–627, San Diego, CA.

Received May 22, 1995; accepted April 8, 1996.