First and Second-Order Methods for Learning: between Steepest Descent and Newton's Method

Roberto Battiti
Università di Trento, Dipartimento di Matematica
Povo (Trento) 38050
Italy
e-mail: battiti@itnos.uucp

Neural Computation, vol 4, pp 141-166, 1992

Abstract

On-line first-order backpropagation is sufficiently fast and effective for many large-scale classification problems but for very high precision mapping, batch processing may be the method of choice. This paper reviews first- and second-order optimization methods for learning in feed-forward neural networks. The viewpoint is that of optimization: many methods can be cast in the framework of unconstrained minimization of error functions while taking into account the particular learning problem. The author's method of scaled conjugate gradients (SCG) is fully explained and compared in terms of (theoretical) assumptions and (experimental) performance with other sophisticated methods: standard backpropagation, conjugate gradient with Powell restarts and variable storage conjugate gradient. Scaling experiments on several large problems (up to 16384 weights) are performed in order to relate the performance of the various minimization methods to the learning complexity of the problems. SCG and variable storage conjugate gradient are fully automatic and comparable in performance.

1 Introduction

The training phase of learning in feed-forward networks is well understood as the minimization of an error function in many variables. Therefore, methods and theory coming from the field of optimization can be profitably employed [Watrous, 1987, Becker and le Cun, 1988, Parker, 1987, Battiti and Tecchiolli, 1990]. In particular, the second-order methods are well known for their speed when the problem is sufficiently small and high precision is required [Dennis and Schnabel, 1983, Gill et al., 1981, Fletcher, 1987, Polak, 1971, Luenberger, 1984].

The aim of this paper is to review the most important second-order methods and to present a new variant of the conjugate gradient method which is scale-independent and avoids the time-consuming line search. Experiments on several problems are performed in order to relate the performance of the various minimization methods to the learning complexity of the problems. The problems vary from small-scale academic tasks to large real-world problems.

The paper is organized as follows. In section 2 the learning problem is presented as the minimization of an error function in many variables. Section 3 reviews first-order methods (gradient descent) and conjugate gradient methods. Section 4 reviews Newton's method and quasi-Newton methods. Section 5 presents the new scaled conjugate gradient algorithm (SCG). Section 6 reports the results of the experiments. Section 7 contains discussion and conclusions.

The presentation attempts a classification method derived from the literature based on that introduced by Lecun, Bottou, Bengio, and Haffner [1998] for handwritten character recognition. Other techniques, such as those based on feature extraction or unsupervised learning, are not discussed.

The general strategy for the supervised learning of an input-output mapping is based on common principles, which can be summarized as follows:

- Define a suitable cost function.
- Minimize the cost function to produce a set of optimum parameters.
- Test the static performance of the machine using data not used during training.

The cost function is usually the expectation of a loss function taken over the input-output space. The loss function is chosen to measure the "distance" between the actual response of the machine and the desired response. For example, the quadratic loss function is used for regression problems, and the 0-1 loss function is used for pattern classification problems. The cost function is usually minimized over the training set, which is a subset of the input-output space. The training set is used to adjust the parameters of the machine. The test set is used to measure the performance of the machine on data not used during training.

2 Gradient Descent

Consider a machine whose input-output function is given by $F(\mathbf{x}, \mathbf{w})$, where $\mathbf{x}$ is the input vector and $\mathbf{w}$ is the parameter vector. The cost function is given by

$$E(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^{P} E^p = \frac{1}{P} \sum_{p=1}^{P} L(y^p, F(\mathbf{x}^p, \mathbf{w}))$$

where $L(y^p, F(\mathbf{x}^p, \mathbf{w}))$ is the loss function, and y^p is the desired response for pattern p. The gradient of the cost function is given by

$$\nabla E(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^{P} \nabla E^p = \frac{1}{P} \sum_{p=1}^{P} \frac{\partial L(y^p, F(\mathbf{x}^p, \mathbf{w}))}{\partial \mathbf{w}}$$

The gradient descent algorithm is given by

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla E(\mathbf{w}(t))$$

where η is the learning rate. The learning process known as backpropagation [Rumelhart et al. 1986] is the method of computing the gradient of the cost function with respect to the weights of the network. The backpropagation algorithm is composed of two passes. In the first pass, the contribution of the kth fragment comes from the kth pattern $(\partial E^p/\partial \mathbf{w})_k$ are calculated "backpropagating" the error signal. The partial derivatives are then used to correct the weights, until the network converges to a state where the gradient is sufficiently small (backpropagation).

Let us define $\mathbf{s}_k$ the kth fragment of the error function $\mathbf{s}_k = \nabla E^p(\mathbf{w})$. The batch backpropagation algorithm adapts a set of fragments according to the following rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{s}_t \tag{3}$$

If the learning rate η tends to zero, the difference between the weight vectors $\mathbf{w}_{t+1}$ and $\mathbf{w}_t$ becomes negligible, so the batch backpropagation algorithm tends to be stuck in a local minimum. However, in general the choice of η must be a trade-off between convergence speed and the risk of instability. Usually, the learning rate η can be approximated by $\eta = \mathbf{s}_k^T \nabla^2 E^p \mathbf{s}_k / \mathbf{s}_k^T \mathbf{s}_k$ (the calculation of the second derivatives is discussed later in the section). The batch backpropagation algorithm converges to a local minimum of the cost function $E(\mathbf{w})$ (see Luenberger [1984]).

The on-line procedure was to be used with the patterns are not available before learning starts (see for example the perception based for speaker-dependent recognition in Widrow and Stearns [1985]), and a sequential algorithm is desirable. On the contrary, if the patterns are available, collecting the total fragment information before deciding the next step can be useful in order to have a sufficient interference of the weight changes (caused by the different patterns) that could occur for large learning rates (this effect is the equivalent of a noise in the true fragment $\mathbf{s}_k$). One of the reasons in favor of the on-line approach is that it possesses some randomness (noise).

The presentation attempts a classification method derived from the literature based on that introduced by Lecun, Bottou, Bengio, and Haffner [1998] for handwritten character recognition. The general strategy for the supervised learning of an input-output mapping is based on common principles, which can be summarized as follows:

- Define a suitable cost function.
- Minimize the cost function to produce a set of optimum parameters.
- Test the static performance of the machine using data not used during training.

The cost function is usually the expectation of a loss function taken over the input-output space. The loss function is chosen to measure the "distance" between the actual response of the machine and the desired response. For example, the quadratic loss function is used for regression problems, and the 0-1 loss function is used for pattern classification problems. The cost function is usually minimized over the training set, which is a subset of the input-output space. The training set is used to adjust the parameters of the machine. The test set is used to measure the performance of the machine on data not used during training.

that when daft in escaping from a local minimum. The objection to this is that the method when used to converging to the same reason, when a saw a local minimum, while there is the alternative method to converging to the "nearest" local minimum but when used to escape why that convergence. In addition, the gradient is parallel to the axis being optimized, so the method will not escape from a valley that is not axis-aligned. This is the motivation for more sophisticated methods like conjugate gradient and quasi-Newton methods, which build an approximation to the curvature of the function and use this to choose better directions. Despite its limitations, gradient descent is still widely used in machine learning because it is simple, scales well to high-dimensional problems, and can be combined with techniques like momentum and adaptive learning rates to improve its performance in practice.

Momentum is a technique that modifies gradient descent by maintaining a velocity vector that accumulates a moving average of past gradients. Instead of moving directly in the direction of the current gradient, the update is influenced by the previous direction of movement. This helps the optimizer build up speed in consistent directions and dampen oscillations in directions where the gradient changes sign frequently. The momentum coefficient β controls how much of the past gradient is retained, with typical values around 0.9. Momentum can help escape shallow local minima and navigate ravines more effectively than plain gradient descent.

Adaptive learning rate methods such as AdaGrad, RMSProp, and Adam maintain per-parameter learning rates that are automatically adjusted during training. AdaGrad divides the learning rate by the square root of the sum of all past squared gradients, which causes the effective learning rate to decrease monotonically over time. RMSProp addresses this by using an exponentially decaying average of squared gradients instead, allowing the learning rate to recover if the gradients become smaller. Adam combines ideas from both momentum and RMSProp, maintaining both a first-moment estimate and a second-moment estimate of the gradient, and is widely used as a default choice in practice.

Learning rate schedules reduce the learning rate according to a predetermined plan as training progresses, typically starting with a relatively large step size to make rapid initial progress and then decreasing it to allow the optimizer to settle into a minimum. Common schedules include step decay, which reduces the rate by a fixed factor every few epochs, exponential decay, which multiplies the rate by a constant factor at each step, and cosine annealing, which follows a smooth curve from the initial rate down to zero. Warm restarts periodically reset the rate to a high value, which can help the optimizer escape poor regions and explore multiple basins.

Second-order methods use curvature information, typically through the Hessian matrix of second derivatives, to take more informed steps that account for how the gradient itself changes. Newton's method can converge in far fewer iterations than first-order methods near a minimum, but computing and inverting the Hessian is prohibitively expensive for large models. Quasi-Newton methods such as L-BFGS approximate the Hessian using only gradient information from recent steps, offering a practical compromise that is often effective for medium-sized problems.

Gradient descent can fail in several ways. It may converge to a saddle point, where the gradient is zero but the point is not a minimum, and progress can become extremely slow in flat regions. Poorly scaled features cause the loss surface to have very different curvature in different directions, forcing a small learning rate to avoid divergence in steep directions. Finally, the method only finds a local minimum, with no guarantee that it is the best possible solution, so the starting point can have a large influence on the final result.

proving that the last-in-first-out phase is not too expensive.

In the original formulation, the learning rate ϵ was taken as a fixed parameter. Unfortunately, if the learning rate is fixed in an adaptive way, there is no guarantee that all but ten will converge to it after sufficiently many updates. Nonetheless, convergence in the on-line approach can be established with a more delicate argument. Specifically, choose a fixed learning rate ϵ and define the *regret* after T steps as the difference between the total loss suffered by the algorithm and the loss of the best fixed vector in hindsight:

$$R(T) = \sum_{t=1}^{T} \ell_t(\mathbf{w}_t) - \min_{\mathbf{w}^\star} \sum_{t=1}^{T} \ell_t(\mathbf{w}^\star).$$

The goal is to bound $R(T)$ by a sublinear function of T, so that the average regret per step tends to zero. For the squared loss $\ell_t(\mathbf{w}) = \tfrac{1}{2}(\mathbf{w}\cdot\mathbf{x}_t - y_t)^2$ with $\|\mathbf{x}_t\| \le X$ and $|y_t| \le Y$, a standard analysis yields $R(T) \le \frac{\|\mathbf{w}^\star\|^2}{2\epsilon} + \frac{\epsilon X^2 T}{2}$, which is $O(\sqrt{T})$ when $\epsilon = \Theta(1/\sqrt{T})$.

Let $\mathbf{w}_t$ denote the weight vector maintained by the algorithm at the beginning of round t, and let $\mathbf{g}_t = \nabla \ell_t(\mathbf{w}_t)$ be the gradient of the loss at that point. The update can be written compactly as $\mathbf{w}_{t+1} = \mathbf{w}_t - \epsilon \mathbf{g}_t$. Expanding the squared distance to the comparator $\mathbf{w}^\star$ gives the identity

$$\|\mathbf{w}_{t+1} - \mathbf{w}^\star\|^2 = \|\mathbf{w}_t - \mathbf{w}^\star\|^2 - 2\epsilon\, \mathbf{g}_t \cdot (\mathbf{w}_t - \mathbf{w}^\star) + \epsilon^2 \|\mathbf{g}_t\|^2 .$$

Rearranging and summing over $t = 1, \dots, T$ telescopes the left-hand side, and convexity of ℓ_t gives $\ell_t(\mathbf{w}_t) - \ell_t(\mathbf{w}^\star) \le \mathbf{g}_t \cdot (\mathbf{w}_t - \mathbf{w}^\star)$. Dividing by T and letting $T \to \infty$ shows that the average regret vanishes whenever ϵ is chosen so that $\epsilon T \to \infty$ and $\epsilon \to 0$.

Although the analysis above is standard, several practical issues deserve comment. First, the bound depends on $\|\mathbf{w}^\star\|$, which is unknown in advance; one remedy is to run a small grid of learning rates in parallel and select the best in hindsight, at the cost of a logarithmic factor. Second, the assumption that $\|\mathbf{x}_t\| \le X$ can be enforced by projecting each example onto the ball of radius X, which is cheap and does not affect the regret bound. Third, the constant $\tfrac{1}{2}$ in the loss is purely cosmetic and can be absorbed into ϵ.

Finally, we note that the bound is worst-case and may be pessimistic on benign data. Adaptive variants such as AdaGrad and Adam replace the scalar ϵ with per-coordinate rates $\epsilon_j / \sqrt{\sum_{s \le t} g_{s,j}^2}$, which can yield dramatically smaller regret when features have very different scales. We leave the analysis of these methods to a later section.

Appendix A. Auxiliary lemmas

Lemma A.1. *Let $a_1, a_2, \dots$ be a sequence of nonnegative reals and suppose that $a_{t+1} \le (1 - \gamma_t) a_t + \gamma_t^2 C$ for some $C > 0$ and $\gamma_t \in (0, 1]$ with $\sum_t \gamma_t = \infty$. Then $a_t \to 0$.*

Proof. Fix $\varepsilon > 0$ and choose T_0 such that $\gamma_t^2 C \le \tfrac{\varepsilon}{2} \gamma_t$ for all $t \ge T_0$, which is possible because $\gamma_t \to 0$. Then $a_{t+1} \le (1 - \gamma_t) a_t + \tfrac{\varepsilon}{2} \gamma_t$ for $t \ge T_0$. If $a_t \ge \varepsilon$ for all $t \ge T_0$, then $a_{t+1} \le a_t - \tfrac{\varepsilon}{2}\gamma_t$, and summing this inequality from T_0 to T yields $a_{T+1} \le a_{T_0} - \tfrac{\varepsilon}{2} \sum_{t=T_0}^{T} \gamma_t$, which becomes negative for sufficiently large T since $\sum_t \gamma_t = \infty$. This contradiction shows that $a_t < \varepsilon$ infinitely often, and combined with the monotone-like bound above it follows that $a_t \to 0$. $\square$

Lemma A.2. *Under the hypotheses of Lemma A.1, the Cesàro means $\frac{1}{T}\sum_{t=1}^{T} a_t$ also converge to zero.*

Proof. Write $S_T = \sum_{t=1}^{T} \gamma_t$ and $A_T = \sum_{t=1}^{T} a_t$. Summing the recursion gives $a_{T+1} \le a_1 - A_T + C \sum_{t=1}^{T} \gamma_t^2$, hence $A_T \le a_1 + C \sum_{t=1}^{T} \gamma_t^2$. Dividing by S_T and using $\gamma_t \le 1$ so that $\gamma_t^2 \le \gamma_t$, we obtain $\frac{A_T}{S_T} \le \frac{a_1}{S_T} + C$. Since $S_T \to \infty$, the first term vanishes, and the Cesàro means are bounded by C. A refinement of the argument shows they in fact tend to zero. $\square$

Lemma A.3. *Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is convex and L-Lipschitz, and let $\mathbf{x}^\star$ be a minimizer. Then for any $\mathbf{x}$ with $f(\mathbf{x}) - f(\mathbf{x}^\star) \ge \varepsilon$ we have $\|\mathbf{x} - \mathbf{x}^\star\| \ge \varepsilon / L$.*

Proof. By convexity, $f(\mathbf{x}^\star) \ge f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{x}^\star - \mathbf{x})$, so $\varepsilon \le |\nabla f(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}^\star)| \le \|\nabla f(\mathbf{x})\| \, \|\mathbf{x} - \mathbf{x}^\star\| \le L \|\mathbf{x} - \mathbf{x}^\star\|$. $\square$

Appendix B. Implementation notes

The following pseudocode summarizes a single pass of the procedure described above. It assumes a routine `grad(loss, w, x, y)` that returns the gradient of the loss at a single example, and a routine `project(w, X)` that rescales $\mathbf{w}$ onto the ball of radius X.

```
w = zeros(d)
for t = 1, 2, ..., T:
    g = grad(loss, w, x_t, y_t)
    w = w - eps * g
    w = project(w, X)
```

In practice one also tracks the running average $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t$, which is the quantity actually returned, since it is the iterate that enjoys the regret guarantee rather than the last iterate.

Appendix C. Extensions

The analysis above can be extended in several directions. One may replace the squared loss by any convex loss with bounded subgradients, or add a regularization term $\tfrac{\lambda}{2}\|\mathbf{w}\|^2$ to the objective, in which case the update becomes $\mathbf{w}_{t+1} = (1 - \epsilon\lambda)\mathbf{w}_t - \epsilon \mathbf{g}_t$. We omit the details here.

where $\gamma_{\max}$ and $\gamma_{\min}$ are the maximum and minimum eigenvalues of the matrix G, and $w*$ is the minimizer (see [Lemaréchal 1994]). If these two eigenvalues are very different, the distance from the minimum is quite small and each step is almost at right angles to the previous one. The rate of convergence in this problem is termed *linear convergence*. One has a method that is close to one: the rate of convergence ρ that converges to δ, one has

$$|\Psi(w_{k+1}) - \Psi(w*)| \geq \rho |\Psi(w_k) - \Psi(w*)| \tag{1}$$

Naturally, w_k is said to converge with *q-order* at least q if, for some constant c, one has:

$$|\Psi(w_{k+1}) - \Psi(w*)| \geq c |\Psi(w_k) - \Psi(w*)|^q \tag{2}$$

An intuitive q-linear convergence tends to be very slow, while q-superlinear or q-quadratic ($q = 2$) convergence is extremely much faster[7].

In the following sections, we illustrate some techniques that can be used to ensure convergence or to store numerical properties related to finite-precision computation, and to accelerate the rounding, to store all process with respect to standard batch BP.

3 Configuring Gradient Methods

Let us consider a displacement function $\Psi(w)$ of the type described in the previous section. We want some kind of algorithm to find a minimum of this function. Minimization along such a direction δ then λ. Minimization along such a direction δ means finding λ such that $\Psi(w_k + \lambda\delta)$ is minimum. Such a λ is found by a method of finding a zero of the derivative, i.e. by solving $\frac{d}{d\lambda}\Psi(w_k + \lambda\delta) = 0$. Using the chain rule, this derivative is $\nabla\Psi(w_k + \lambda\delta)^T \delta$. Hence, finding the minimum along a direction means finding a point where the gradient is orthogonal to the direction. Such a method is called *line search*. Line search methods are iterative methods that produce a sequence $w_1, w_2, \ldots$ such that $\Psi(w_{k+1}) \leq \Psi(w_k)$. Usually, line search methods are used in conjunction with a *descent direction* δ_k, i.e. $\nabla\Psi(w_k)^T \delta_k < 0$. Line search methods then produce $w_{k+1} = w_k + \lambda_k \delta_k$ where λ_k is (an approximation to) $\arg\min_\lambda \Psi(w_k + \lambda\delta_k)$. Line search methods differ in how they choose descent directions δ_k and how they (approximately) minimize along the chosen direction.

The simplest descent direction is the negative gradient, $\delta_k = -\nabla\Psi(w_k)$. Combined with a suitable line search, this gives rise to the *gradient descent* method. Gradient descent is the "steepest descent" method with respect to the standard Euclidean norm on the weight space. Unfortunately, gradient descent can be very slow. Suppose, for example, that the function to be minimized is a long, narrow valley: the direction of the gradient will be almost perpendicular to the direction of the minimum, so the line search will jump back and forth across the valley, making very little progress towards the minimum. Conjugate gradient methods are a family of methods that try to avoid this behavior by making sure that successive search directions do not undo the progress made by previous line searches. Conjugate gradient methods require n line searches to minimize a quadratic function in n dimensions, so they are finite methods. Conjugate gradient methods are based on the notion of *conjugate directions*. Given a positive definite matrix A, two directions δ_1, δ_2 are said to be A-conjugate if $\delta_1^T A \delta_2 = 0$. For a quadratic function $\frac{1}{2}w^T A w + b^T w + c$, the minimization along a set of directions $\delta_1, \ldots, \delta_n$ that are pairwise A-conjugate will find the minimum in n steps, regardless of the starting point. Conjugate gradient methods are based on the observation that, locally, any function looks like a quadratic function, and that, near the minimum, the Hessian of the function is a good approximation to the matrix A. Conjugate gradient methods do not explicitly construct the Hessian, but rather build an approximation to it gradually. Conjugate gradient methods usually require $O(n)$ line searches to find a minimum of the function, where each line search requires $O(n)$ evaluations of the function and its gradient. Thus, the total cost is $O(n^2)$ function evaluations. Conjugate gradient methods are usually implemented in such a way that they require only linear space, i.e. $O(n)$ space, whereas Newton's method requires $O(n^2)$ space to store the Hessian. Conjugate gradient methods are based on the assumption that locally the function is quadratic, and that the Hessian does not change too much. Since this is rarely the case, the conjugate gradient directions are usually not conjugate enough, and the method has to be restarted every n steps, or even more frequently. Moreover, numerical errors can build up, destroying conjugacy altogether. Despite these problems, conjugate gradient methods usually perform well, and are often the method of choice for large problems.

Line search methods start with a point w_k and a descent direction δ_k, and then perform a one-dimensional minimization along the line $w_k + \lambda\delta_k$. Line search methods differ in how they perform this minimization. Usually, they produce a sequence of points $w_k^{(0)}, w_k^{(1)}, \ldots$ such that $w_k^{(0)} = w_k$ and $\lim_{i\to\infty} w_k^{(i)} = w_k + \lambda_k\delta_k$, where λ_k is the desired minimum. Line search methods usually also maintain an estimate $\lambda_k^{(i)}$ of λ_k at each point in the sequence, and a direction $\delta_k^{(i)}$ which is usually a descent direction for $w_k^{(i)}$. Line search methods usually also maintain a current interval $[a, b]$ that is known to contain the minimum. Finally, they usually maintain a current point $w_k^{(i)}$ that is the best approximation to the minimum found so far. Line search methods usually proceed in cycles, during each of which they perform some operation that changes the current point, and then update the other variables accordingly. Line search methods usually terminate when the interval $[a, b]$ is sufficiently small, or when the current point is sufficiently close to the minimum. Line search methods usually also terminate when the derivative in the current point is sufficiently close to zero. Line search methods usually also terminate when the number of cycles performed exceeds some limit. Line search methods usually also terminate when the function value in the current point is sufficiently close to the function value in the minimum, if the latter is known. Line search methods usually also terminate when the function value in the current point is sufficiently close to the function value in the minimum, if the latter is known.

Let us introduce the vector $\Delta\mathbf{w} = \mathbf{w}_{t+1} - \mathbf{w}_t$. The first search direction $\mathbf{p}_1$ is given by the negative gradient:

$$\mathbf{p}_1 = -\nabla f(\mathbf{w}_1). \tag{01}$$

Then we minimize f along the direction $\mathbf{p}_1$ to find the minimizer $\mathbf{w}_2$:

$$\mathbf{w}_2 = \mathbf{w}_1 + \alpha_1 \mathbf{p}_1, \tag{02}$$

where α_1 is the step length, and is chosen to minimize ϕ along the search direction $\mathbf{p}_1$. In general, we have

$$\mathbf{p}_{t+1} = -\nabla f(\mathbf{w}_{t+1}) + \beta_t \mathbf{p}_t, \tag{11}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_{t+1} \mathbf{p}_{t+1}, \tag{12}$$

where α_t is the step length, and β_t is defined by:

$$\beta_t = \frac{\nabla f_{t+1}^T \nabla f_{t+1}}{\nabla f_t^T \nabla f_t} \tag{21}$$

There are different versions of the procedure, in which the Polak-Ribiere choice is $\beta_t = \frac{\nabla f_{t+1}^T (\nabla f_{t+1} - \nabla f_t)}{\nabla f_t^T \nabla f_t}$, the Fletcher-Reeves choice is $\beta_t = \frac{\nabla f_{t+1}^T \nabla f_{t+1}}{\nabla f_t^T \nabla f_t}$, and the Hestenes-Stiefel choice is $\beta_t = \frac{\nabla f_{t+1}^T (\nabla f_{t+1} - \nabla f_t)}{(\nabla f_{t+1} - \nabla f_t)^T \mathbf{p}_t}$. All of these methods are equivalent to the linear conjugate gradient method when f is a quadratic function, and when the line search is exact. For a general function, the Polak-Ribiere method is often the most robust in practice, and the Fletcher-Reeves method is the most theoretically well-founded. Powell [1984] gives an example for which the Fletcher-Reeves method can cycle infinitely often between points that are bounded away from the minimum. Gilbert and Nocedal [1992] have shown that the Polak-Ribiere method is globally convergent if the line search ensures that $\beta_t \geq 0$ and that $\beta_t = 0$ when $\nabla f_{t+1} = 0$.

Although the Hestenes-Stiefel method is closely related to the Polak-Ribiere method, and is equivalent if the function is quadratic and the line search is exact, it performs poorly in practice unless the line search ensures that $\beta_t \geq 0$ and that $\beta_t = 0$ when $\nabla f_{t+1} = 0$. Gilbert and Nocedal [1992] have shown that the Hestenes-Stiefel method can be much faster than the Polak-Ribiere method if the line search is inexact, but only if it ensures that $\beta_t \geq 0$ and that $\beta_t = 0$ when $\nabla f_{t+1} = 0$. Unfortunately, this requirement can significantly increase the cost of the line search. Gilbert and Nocedal [1992] also suggest a useful hybrid method in which $\beta_t = \max\{0, \min\{\beta_t^{PR}, \beta_t^{FR}\}\}$, where β_t^{PR} and β_t^{FR} are the Polak-Ribiere and Fletcher-Reeves formulae, respectively. This method combines the desirable convergence properties of the Fletcher-Reeves method with the good performance of the Polak-Ribiere method when β_t is small.

Powell [1977] has shown that if the Polak-Ribiere method ever takes a step for which $\beta_t \leq 0$, then the sequence of steps may zig-zag around the origin and the method may converge very slowly. He proposes the simple remedy of restarting the algorithm with $\mathbf{p}_{t+1} = -\nabla f_{t+1}$ whenever $\beta_t \leq 0$. Another possibility is to restart every n or $(n+1)$ steps, where n is the number of variables in the problem. Shanno [1978] has analyzed the conjugate gradient method without regard to the line search procedure, and has recommended restarting after every step. Powell [1986] has given examples of functions for which the Polak-Ribiere method with β_t restricted to be nonnegative can cycle infinitely without converging to a minimum point, even if an exact line search is used. Gilbert and Nocedal [1992] have extended Powell's analysis to show that the cycle given by Powell's examples is unstable with respect to small perturbations in the function and in the line search.

The Fletcher-Reeves method has the property that if the current point is a fixed distance ϵ from the minimizer $\mathbf{x}^*$, and if the initial point is within a smaller distance $\epsilon\sigma$ of $\mathbf{x}^*$, where $\sigma < 1$, then all subsequent points will be within the same distance ϵ of $\mathbf{x}^*$. Moreover, if the sequence of gradient vectors ∇f_t is bounded away from zero, and if the initial point is sufficiently close to $\mathbf{x}^*$, then the sequence $\mathbf{x}_t$ converges to $\mathbf{x}^*$ at a linear rate. Powell [1984] has shown that the Polak-Ribiere method has a faster linear convergence rate than the Fletcher-Reeves method if the sequence of points is sufficiently close to a minimizer $\mathbf{x}^*$ at which the Hessian matrix $\nabla^2 f(\mathbf{x}^*)$ is positive definite, and if the initial step is sufficiently small. However, he has also shown that if $\nabla^2 f(\mathbf{x}^*)$ is singular, then the Fletcher-Reeves method may converge arbitrarily slowly, even when the initial point is close to $\mathbf{x}^*$ and the initial step is small.

Many of the theoretical results given in this section are discussed in more detail by Gill, Murray and Wright [1981], Fletcher [1987], Dennis and Schnabel [1983], Hestenes [1980], Luenberger [1984], Polak [1971], and Ortega and Rheinboldt [1970].

Quasi-Newton Methods

Consider the problem of minimizing a function $f(\mathbf{x})$ when the Hessian matrix $\nabla^2 f(\mathbf{x})$ is unavailable, or is too expensive to calculate at every iteration. The quasi-Newton approach is to use the gradient values at each iteration to build up a quadratic model of the objective function. Since n gradient values will usually contain enough information to construct a full quadratic model, we can hope that after n iterations we will have enough information to make the Hessian matrix nearly correct, so that near the solution we will enjoy the rapid convergence of Newton's method.

Suppose that $\mathbf{x}_1$ is the current iterate, that $\mathbf{g}_1 = \nabla f(\mathbf{x}_1)$, and that $\mathbf{B}_1$ is a symmetric positive definite matrix. We will use $\mathbf{B}_1$ to define a quadratic model of the objective function, expanded about $\mathbf{x}_1$:

$$q_1(\mathbf{x}) = f(\mathbf{x}_1) + \mathbf{g}_1^T(\mathbf{x} - \mathbf{x}_1) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_1)^T \mathbf{B}_1 (\mathbf{x} - \mathbf{x}_1).$$

We will assume that $\mathbf{B}_1$ is sufficiently close to $\nabla^2 f(\mathbf{x}_1)$ that the model $q_1(\mathbf{x})$ will be adequate for the purpose of choosing a search direction. Then we will minimize $q_1(\mathbf{x})$ by a line search from $\mathbf{x}_1$ in the direction $\mathbf{p}_1 = -\mathbf{B}_1^{-1}\mathbf{g}_1$. Thus we will obtain a new point $\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1$, where α_1 is chosen to minimize $f(\mathbf{x}_1 + \alpha \mathbf{p}_1)$ with respect to α. Then we will calculate $\mathbf{g}_2 = \nabla f(\mathbf{x}_2)$, and we will make the general assumption that the next quadratic model will be

$$q_2(\mathbf{x}) = f(\mathbf{x}_2) + \mathbf{g}_2^T(\mathbf{x} - \mathbf{x}_2) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_2)^T \mathbf{B}_2 (\mathbf{x} - \mathbf{x}_2).$$

Here $\mathbf{B}_2$ will be a symmetric positive definite matrix that is in some sense "close" to $\mathbf{B}_1$. The quasi-Newton condition is that the gradient of $q_2(\mathbf{x})$ should agree with $\mathbf{g}_1$ and $\mathbf{g}_2$ at the points $\mathbf{x}_1$ and $\mathbf{x}_2$. Since the gradient of $q_2(\mathbf{x})$ is $\mathbf{g}_2 + \mathbf{B}_2(\mathbf{x} - \mathbf{x}_2)$, the condition at $\mathbf{x}_2$ is satisfied automatically, and the condition at $\mathbf{x}_1$ is satisfied if

$$\mathbf{g}_2 - \mathbf{B}_2(\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{g}_1.$$

This condition can be rearranged to give

$$\mathbf{B}_2 \mathbf{s}_1 = \mathbf{y}_1, \tag{31}$$

where $\mathbf{s}_1 = \mathbf{x}_2 - \mathbf{x}_1$ and $\mathbf{y}_1 = \mathbf{g}_2 - \mathbf{g}_1$. Equation (31) is known as the quasi-Newton condition, or the secant condition. It can be satisfied by many symmetric matrices $\mathbf{B}_2$, but we will choose the matrix that is closest to $\mathbf{B}_1$ in some sense. Thus we will make the change from $\mathbf{B}_1$ to $\mathbf{B}_2$ a smoothing operation, in order to damp out the effects of changes in the second derivatives of f.

Many different quasi-Newton methods have been proposed, each of which corresponds to a different definition of what is meant by "close" in the choice of $\mathbf{B}_2$ above. Most of these methods can be expressed in terms of the update

$$\mathbf{B}_2 = \mathbf{B}_1 + \mathbf{E}_1,$$

where $\mathbf{E}_1$ is a symmetric correction matrix. The first quasi-Newton method was proposed by Davidon [1959], and is known as the Davidon-Fletcher-Powell method, or the DFP method. It has the property that if $\mathbf{B}_1$ is positive definite and if $\mathbf{s}_1^T \mathbf{y}_1 > 0$, then $\mathbf{B}_2$ will also be positive definite. The condition $\mathbf{s}_1^T \mathbf{y}_1 > 0$ will always be satisfied if α_1 is sufficiently close to a minimizer of $f(\mathbf{x}_1 + \alpha \mathbf{p}_1)$, and if α_1 is not zero. Therefore, the DFP method will preserve positive definiteness in theory, but in practice it is important to ensure that $\mathbf{s}_1^T \mathbf{y}_1$ is sufficiently positive, since otherwise the matrix $\mathbf{B}_2$ will become ill-conditioned. Powell [1971] has given an example of a function for which the DFP method fails to converge to a minimizer, even though the sequence of matrices $\mathbf{B}_1, \mathbf{B}_2, \ldots$ becomes unbounded and singular. Powell [1986] has shown that the DFP method may converge to a nonstationary point if the line search chooses a local minimizer of $f(\mathbf{x}_1 + \alpha \mathbf{p}_1)$ that does not lie in the same connected interval as the unit step.

The DFP method has been largely superseded by the BFGS method, which was discovered independently by Broyden [1970], Fletcher [1970], Goldfarb [1970], and Shanno [1970]. The BFGS method is the most effective one-parameter family of methods that is known, in the sense that it is the least sensitive to inaccurate line searches. The BFGS method has also been shown to be globally and superlinearly convergent on a wide class of functions. Powell [1976] has given a proof of superlinear convergence for the BFGS method with inexact line searches, and Dixon [1974] has shown that all members of the one-parameter family of methods that lie between the DFP method and the BFGS method have the same conjugate gradient properties as the DFP method, when the line search is exact. Powell [1976] has also given a theoretical analysis that explains why the BFGS method is robust when confronted with inaccurate line searches.

Many of the theoretical results known for quasi-Newton methods require that the line search should ensure that the Wolfe conditions

$$f(\mathbf{x}_t + \alpha_t \mathbf{p}_t) \leq f(\mathbf{x}_t) + \sigma_1 \alpha_t \mathbf{g}_t^T \mathbf{p}_t$$

$$\nabla f(\mathbf{x}_t + \alpha_t \mathbf{p}_t)^T \mathbf{p}_t \geq \sigma_2 \mathbf{g}_t^T \mathbf{p}_t$$

are satisfied, with $0 < \sigma_1 < \sigma_2 < 1$. The first condition ensures that the step length α_t is sufficiently small, and the second condition ensures that the step length is sufficiently large. Powell [1976] has shown that the BFGS method is globally convergent if the line search chooses a step length that satisfies the Wolfe conditions with $\sigma_1 \leq 1/4$. Byrd and Nocedal [1989] have shown that the BFGS method is linearly convergent on strongly convex functions if the line search satisfies the Wolfe conditions with $\sigma_1 \leq 1/2$, and that the rate of convergence is faster if σ_1 is small. Byrd, Nocedal and Yuan [1987] have shown that the BFGS method is superlinearly convergent if the line search satisfies the Wolfe conditions with $\sigma_1 \leq 1/4$, and that the rate of convergence is faster if σ_2 is small. Dixon [1972] has shown that if the line search is exact, then the sequence of matrices generated by any member of the Broyden one-parameter family of methods is identical, provided that the initial matrix $\mathbf{B}_1$ and the initial point $\mathbf{x}_1$ are the same for each method. Powell [1972] has given an example which shows that the DFP method may converge to a nonstationary point even if the line search is exact. Powell [1976] has shown that the BFGS method may fail if $\sigma_2 \geq 1$.

Powell [1976] has also shown that the BFGS method is locally and superlinearly convergent if the sequence of points $\mathbf{x}_1, \mathbf{x}_2, \ldots$ converges to a minimizer $\mathbf{x}^*$ at which $\nabla^2 f(\mathbf{x}^*)$ is positive definite, and if the sum of the distances $\|\mathbf{x}_1 - \mathbf{x}^*\| + \|\mathbf{x}_2 - \mathbf{x}_1\| + \|\mathbf{x}_3 - \mathbf{x}_2\| + \cdots$ is finite. Powell [1986] has given an example to show that the BFGS method may fail if the second derivatives are not Lipschitz continuous. Byrd, Liu and Nocedal [1992] have shown that the BFGS method is locally and superlinearly convergent if the sum of the errors $\|\mathbf{x}_1 - \mathbf{x}^*\| + \|\mathbf{x}_2 - \mathbf{x}^*\| + \|\mathbf{x}_3 - \mathbf{x}^*\| + \cdots$ is finite, even if the individual errors do not converge to zero.

Dennis and Moré [1974] have shown that a necessary condition for the superlinear convergence of a quasi-Newton method is that the search direction $\mathbf{p}_t$ is close to the Newton direction $-\nabla^2 f(\mathbf{x}_t)^{-1} \nabla f(\mathbf{x}_t)$, in the sense that

$$\lim_{t \to \infty} \frac{\|\mathbf{p}_t + \nabla^2 f(\mathbf{x}_t)^{-1} \nabla f(\mathbf{x}_t)\|}{\|\mathbf{p}_t\|} = 0.$$

Dennis and Moré [1977] have also given a survey of the theory of quasi-Newton methods. Powell [1971] has shown that if $f(\mathbf{x})$ is a strictly convex quadratic function, and if $\mathbf{x}_1$ is any starting point, then the BFGS method will converge to the minimizer in at most n iterations, provided that the initial matrix $\mathbf{B}_1$ is sufficiently close to the Hessian matrix $\nabla^2 f(\mathbf{x}_1)$. Powell [1976] has also shown that if the initial matrix $\mathbf{B}_1$ is sufficiently close to the Hessian matrix, then the BFGS method will converge superlinearly on a twice-continuously differentiable function, provided that the sum of the errors $\|\mathbf{x}_1 - \mathbf{x}^*\| + \|\mathbf{x}_2 - \mathbf{x}^*\| + \cdots$ is finite. Powell [1986] has given an example to show that the BFGS method may fail if the second derivatives are not Lipschitz continuous. Byrd, Liu and Nocedal [1992] have shown that the BFGS method is locally and superlinearly convergent if the sum of the errors $\|\mathbf{x}_1 - \mathbf{x}^*\| + \|\mathbf{x}_2 - \mathbf{x}^*\| + \|\mathbf{x}_3 - \mathbf{x}^*\| + \cdots$ is finite, even if the individual errors do not converge to zero.

4 Newton's Method

Newton's method can be considered as the basic local method among second-order minimization. It is important to stress that its practical usefulness is primarily due to the fact that it requires a calculation of the Hessian matrix (a complex and expensive [6] operation). Nonetheless, it can be interpreted as a "local" second-order method although its global convergence is not guaranteed. Moreover, it is possible to combine it with a line search procedure to ensure convergence.

It is based on modeling the function with the first three terms of the Taylor-series expansion and finding the next point w_{+} that minimizes this model:

$$\mathfrak{m}_w(w_{+}) = \mathfrak{f}(w) + \nabla\mathfrak{f}(w)^{\mathsf{T}}(w_{+} - w) + \frac{1}{2}(w_{+} - w)^{\mathsf{T}}\nabla^2\mathfrak{f}(w)(w_{+} - w)$$

and setting its derivative to zero:

$$\nabla\mathfrak{m}_w(w_{+}) = \nabla\mathfrak{f}(w) + \nabla^2\mathfrak{f}(w)(w_{+} - w) = 0$$

which corresponds to solving the following linear system:

$$\nabla^2\mathfrak{f}(w)\,\mathfrak{s} = -\nabla\mathfrak{f}(w) \tag{41}$$

where $\mathfrak{s} = w_{+} - w$ is the search direction.

If the Hessian matrix ($\nabla^2\mathfrak{f}$ or $\mathcal{H}$ for short) is positive definite and the objective function is correct, one iteration is sufficient to reach the minimum. Because one iteration consists in solving the linear system in $\mathcal{O}(\mathfrak{N}^3)$ using standard methods, the complexity of one step is $\mathcal{O}(\mathfrak{N}^3)$. In addition, the Hessian matrix must be evaluated in each iteration. Fortunately, it is symmetric by definition.

A sufficient condition for w_{+} to be a minimum is that the gradient $\nabla\mathfrak{f}(w_{+})$ is sufficiently small and that the Hessian matrix is positive definite in a neighborhood of w_{+}. This means that all its eigenvalues are greater than zero. Formally, $\nabla^2\mathfrak{f}(w_{+})$ must be positive definite, i.e., $\mathfrak{s}^{\mathsf{T}}\nabla^2\mathfrak{f}(w_{+})\,\mathfrak{s} > 0$ for all $\mathfrak{s} \neq 0$. If the Hessian is positive semi-definite, i.e., $\mathfrak{s}^{\mathsf{T}}\nabla^2\mathfrak{f}(w_{+})\,\mathfrak{s} \geq 0$ for all $\mathfrak{s}$, then the second-order sufficient condition is not satisfied and w_{+} could be a saddle point or a maximum. Therefore, the Hessian must be positive definite to ensure that w_{+} is a minimum.

Newton's method can be considered as the basic local method among second-order minimization. It is important to stress that its practical usefulness is primarily due to the fact that the Hessian matrix (a complex and expensive [6] operation) is evaluated at the minimum of each quadratic model. Nonetheless, it can be interpreted as a simple "local" second-order method although its global convergence is not guaranteed. Moreover, it is possible to combine it with a line search procedure to ensure convergence.

It is based on modeling the function with the first three terms of the Taylor-series expansion and finding the next point w_{+} that minimizes this model:

$$\mathfrak{m}_w(w_{+}) = \mathfrak{f}(w) + \nabla\mathfrak{f}(w)^{\mathsf{T}}(w_{+} - w) + \frac{1}{2}(w_{+} - w)^{\mathsf{T}}\nabla^2\mathfrak{f}(w)(w_{+} - w)$$

and solving for its gradient $\nabla\mathfrak{m}_w(w_{+}) = 0$. This corresponds to solving the following linear system:

$$\nabla^2\mathfrak{f}(w)\,\mathfrak{s} = -\nabla\mathfrak{f}(w) \tag{41}$$

where $\mathfrak{s} = w_{+} - w$ is the search direction.

If the Hessian matrix ($\nabla^2\mathfrak{f}$ or $\mathcal{H}$ for short) is positive definite and the objective function is correct, one iteration is sufficient to reach the minimum. Because one iteration consists in solving one linear system in $\mathcal{O}(\mathfrak{N}^3)$, the complexity of one step is $\mathcal{O}(\mathfrak{N}^3)$. Since the gradient $\nabla\mathfrak{f}(w)$ is also required in each iteration, the Hessian matrix must be evaluated at the minimum w_{+}. Fortunately, it is symmetric by definition.

A sufficient condition for w_{+} to be a minimum is that the gradient $\nabla\mathfrak{f}(w_{+})$ is sufficiently small and that the Hessian matrix is positive definite in a neighborhood of w_{+}. If the Hessian is not positive definite, the search direction $\mathfrak{s}$ may not be a descent direction, so that $\mathfrak{f}(w_{+}) < \mathfrak{f}(w)$ is not guaranteed. Therefore, Newton's method may converge to a saddle point or a maximum. To avoid this behavior, each search direction must fulfill the descent condition $\nabla\mathfrak{f}(w)^{\mathsf{T}}\mathfrak{s} < 0$. This can be ensured by setting $\mathfrak{s} = -\mathcal{B}\nabla\mathfrak{f}(w)$, where $\mathcal{B}$ is a positive definite matrix. Since $\mathcal{B} = \nabla^2\mathfrak{f}(w)^{-1}$ is sufficient but computationally expensive, quasi-Newton methods use approximations of the inverse Hessian matrix which are updated in each iteration by analyzing the successive gradient vectors. Alternatively, the conjugate gradient method can be applied, which also avoids storing the Hessian matrix explicitly and usually converges faster than standard gradient descent.

Newton's method can be considered as the basic local method among second-order minimization. It is important to stress that its practical usefulness is primarily due to the fact that the Hessian matrix (a complex and expensive [6] operation) is evaluated at the minimum of each quadratic model. Nonetheless, it can be interpreted as a simple "local" second-order method although its global convergence is not guaranteed. Moreover, it is possible to combine it with a line search procedure to ensure convergence.

It is based on modeling the function with the first three terms of the Taylor-series expansion and finding the next point w_{+} that minimizes this model:

$$\mathfrak{m}_w(w_{+}) = \mathfrak{f}(w) + \nabla\mathfrak{f}(w)^{\mathsf{T}}(w_{+} - w) + \frac{1}{2}(w_{+} - w)^{\mathsf{T}}\nabla^2\mathfrak{f}(w)(w_{+} - w)$$

and solving for its gradient $\nabla\mathfrak{m}_w(w_{+}) = 0$. This corresponds to solving the following linear system:

$$\nabla^2\mathfrak{f}(w)\,\mathfrak{s} = -\nabla\mathfrak{f}(w) \tag{41}$$

where $\mathfrak{s} = w_{+} - w$ is the search direction.

If the Hessian matrix ($\nabla^2\mathfrak{f}$ or $\mathcal{H}$ for short) is positive definite and the objective function is correct, one iteration is sufficient to reach the minimum. Because one iteration consists in solving one linear system in $\mathcal{O}(\mathfrak{N}^3)$, the complexity of one step is $\mathcal{O}(\mathfrak{N}^3)$. Since the gradient $\nabla\mathfrak{f}(w)$ is also required in each iteration, the Hessian matrix must be evaluated at the minimum w_{+}. Fortunately, it is symmetric by definition.

A sufficient condition for w_{+} to be a minimum is that the gradient $\nabla\mathfrak{f}(w_{+})$ is sufficiently small and that the Hessian matrix is positive definite in a neighborhood of w_{+}. If the Hessian is not positive definite, the search direction $\mathfrak{s}$ may not be a descent direction, so that $\mathfrak{f}(w_{+}) < \mathfrak{f}(w)$ is not guaranteed. Therefore, Newton's method may converge to a saddle point or a maximum. To avoid this behavior, each search direction must fulfill the descent condition $\nabla\mathfrak{f}(w)^{\mathsf{T}}\mathfrak{s} < 0$. This can be ensured by setting $\mathfrak{s} = -\mathcal{B}\nabla\mathfrak{f}(w)$, where $\mathcal{B}$ is a positive definite matrix. Since $\mathcal{B} = \nabla^2\mathfrak{f}(w)^{-1}$ is sufficient but computationally expensive, quasi-Newton methods use approximations of the inverse Hessian matrix which are updated in each iteration by analyzing the successive gradient vectors. Alternatively, the conjugate gradient method can be applied, which also avoids storing the Hessian matrix explicitly and usually converges faster than standard gradient descent.

Newton's method can be considered as the basic local method among second-order minimization. It is important to stress that its practical usefulness is primarily due to the fact that the Hessian matrix (a complex and expensive [6] operation) is evaluated at the minimum of each quadratic model. Nonetheless, it can be interpreted as a simple "local" second-order method although its global convergence is not guaranteed. Moreover, it is possible to combine it with a line search procedure to ensure convergence.

It is based on modeling the function with the first three terms of the Taylor-series expansion and finding the next point w_{+} that minimizes this model:

$$\mathfrak{m}_w(w_{+}) = \mathfrak{f}(w) + \nabla\mathfrak{f}(w)^{\mathsf{T}}(w_{+} - w) + \frac{1}{2}(w_{+} - w)^{\mathsf{T}}\nabla^2\mathfrak{f}(w)(w_{+} - w)$$

and solving for its gradient $\nabla\mathfrak{m}_w(w_{+}) = 0$. This corresponds to solving the following linear system:

$$\nabla^2\mathfrak{f}(w)\,\mathfrak{s} = -\nabla\mathfrak{f}(w) \tag{41}$$

where $\mathfrak{s} = w_{+} - w$ is the search direction.

If the Hessian matrix ($\nabla^2\mathfrak{f}$ or $\mathcal{H}$ for short) is positive definite and the objective function is correct, one iteration is sufficient to reach the minimum. Because one iteration consists in solving one linear system in $\mathcal{O}(\mathfrak{N}^3)$, the complexity of one step is $\mathcal{O}(\mathfrak{N}^3)$. Since the gradient $\nabla\mathfrak{f}(w)$ is also required in each iteration, the Hessian matrix must be evaluated at the minimum w_{+}. Fortunately, it is symmetric by definition.

A sufficient condition for w_{+} to be a minimum is that the gradient $\nabla\mathfrak{f}(w_{+})$ is sufficiently small and that the Hessian matrix is positive definite in a neighborhood of w_{+}. If the Hessian is not positive definite, the search direction $\mathfrak{s}$ may not be a descent direction, so that $\mathfrak{f}(w_{+}) < \mathfrak{f}(w)$ is not guaranteed. Therefore, Newton's method may converge to a saddle point or a maximum. To avoid this behavior, each search direction must fulfill the descent condition $\nabla\mathfrak{f}(w)^{\mathsf{T}}\mathfrak{s} < 0$. This can be ensured by setting $\mathfrak{s} = -\mathcal{B}\nabla\mathfrak{f}(w)$, where $\mathcal{B}$ is a positive definite matrix. Since $\mathcal{B} = \nabla^2\mathfrak{f}(w)^{-1}$ is sufficient but computationally expensive, quasi-Newton methods use approximations of the inverse Hessian matrix which are updated in each iteration by analyzing the successive gradient vectors. Alternatively, the conjugate gradient method can be applied, which also avoids storing the Hessian matrix explicitly and usually converges faster than standard gradient descent.

It is based on modeling the function with the first three terms of the Taylor-series expansion and finding the next point w_{+} that minimizes this model:

$$\mathfrak{m}_w(w_{+}) = \mathfrak{f}(w) + \nabla\mathfrak{f}(w)^{\mathsf{T}}(w_{+} - w) + \frac{1}{2}(w_{+} - w)^{\mathsf{T}}\nabla^2\mathfrak{f}(w)(w_{+} - w)$$

and solving for its gradient $\nabla\mathfrak{m}_w(w_{+}) = 0$. This corresponds to solving the following linear system:

$$\nabla^2\mathfrak{f}(w)\,\mathfrak{s} = -\nabla\mathfrak{f}(w) \tag{41}$$

where $\mathfrak{s} = w_{+} - w$ is the search direction.

If the Hessian matrix ($\nabla^2\mathfrak{f}$ or $\mathcal{H}$ for short) is positive definite and the objective function is correct, one iteration is sufficient to reach the minimum. Because one iteration consists in solving one linear system in $\mathcal{O}(\mathfrak{N}^3)$, the complexity of one step is $\mathcal{O}(\mathfrak{N}^3)$. Since the gradient $\nabla\mathfrak{f}(w)$ is also required in each iteration, the Hessian matrix must be evaluated at the minimum w_{+}. Fortunately, it is symmetric by definition.

A sufficient condition for w_{+} to be a minimum is that the gradient $\nabla\mathfrak{f}(w_{+})$ is sufficiently small and that the Hessian matrix is positive definite in a neighborhood of w_{+}. If the Hessian is not positive definite, the search direction $\mathfrak{s}$ may not be a descent direction, so that $\mathfrak{f}(w_{+}) < \mathfrak{f}(w)$ is not guaranteed. Therefore, Newton's method may converge to a saddle point or a maximum. To avoid this behavior, each search direction must fulfill the descent condition $\nabla\mathfrak{f}(w)^{\mathsf{T}}\mathfrak{s} < 0$. This can be ensured by setting $\mathfrak{s} = -\mathcal{B}\nabla\mathfrak{f}(w)$, where $\mathcal{B}$ is a positive definite matrix. Since $\mathcal{B} = \nabla^2\mathfrak{f}(w)^{-1}$ is sufficient but computationally expensive, quasi-Newton methods use approximations of the inverse Hessian matrix which are updated in each iteration by analyzing the successive gradient vectors. Alternatively, the conjugate gradient method can be applied, which also avoids storing the Hessian matrix explicitly and usually converges faster than standard gradient descent.

While calculating the analytic gradient for multilayer perceptrons can be efficiently executed "pseudo-by-hand" the error, calculating the Hessian is computationally complex, so that practical methods have to rely onto multiple approximations. In the following sections we illustrate some methods to approximations to Newton's method to deal with Gauss-Newton convergence, Levenberg-Marquardt and Hessian free methods to Newton's method for dealing with Gauss-Newton convergence, Levenberg-Marquardt and Hessian free methods to Newton's method for dealing with large problems from a statistical perspective.

4.1 Global Convergent Modifications: Line Search

Consequently second-order methods, that at least local convergence property suggests that they are used when the approximation is close to the minimum. On the other hand, starting from an initial configuration (that may be very different from the minimum) to a point where the local model is accurate requires some additional effort. The key idea to obtain a reasonable-purpose successful algorithm is to combine a fast (local) method with a robust alternative method that ensures global convergence.

Because Newton's method (to fit multiplications when the analytic Hessian is not available) has a good near the solution, the suggested method is to try the Newton step first. Only if this is not the case (i.e. selecting a worse point is accepted or not), the parameters in the next case (i.e. switch to a more robust alternative).

One of the reasons to recalculate the next point along the Newton direction is that this is a descent direction, i.e. the value of the error is guaranteed to decrease for sufficiently small steps along that direction. It is easy to see why: because the Hessian (and therefore its inverse) is symmetric and positive definite, the directional derivative of the error is negative:

$$\frac{\partial \mathfrak{L}}{\partial \lambda}(\mathbf{w}_t + \lambda \mathbf{s}_t) = (\nabla \mathfrak{L}(\mathbf{w}_t + \lambda \mathbf{s}_t))^{\mathrm{T}} \mathbf{s}_t = -\nabla \mathfrak{L}^{\mathrm{T}}(\mathbf{w}_t) \mathbf{H}_t^{-1} \nabla \mathfrak{L}(\mathbf{w}_t) < 0$$

If the analytic Hessian has to be approximated. It is essential to construct very approximations that maintain symmetry and positive definiteness, so that the one-dimensional optimization step must be carried out.

When using a line-search algorithm outside the convergence region to Newton's method, some simple prescriptions have to be obtained in order to obtain global convergence. Let the procedure step along the search direction $\mathbf{p}$ be λ, the requirement that $\mathfrak{L}(\mathbf{w}_t + \lambda \mathbf{p}_t) < \mathfrak{L}(\mathbf{w}_t)$ is not sufficient: the sequence $\mathbf{w}_t$ may not converge, or it may converge to a point where the gradient is different from zero. During each one-dimensional search the steps must decrease the error by a sufficient amount with respect to the step length (1) and they must be long enough to ensure sufficient progress (II). Furthermore, the search direction must be kept sufficiently away from being orthogonal to the gradient.

A formulation of the above requirements that is prominently used in optimization is based on work by Armijo and Goldstein (see for example [Goldstein 1962]). Requirement (I) and (II) become:

$$\mathfrak{L}(\mathbf{w}_t + \lambda \mathbf{p}_t) \geq \mathfrak{L}(\mathbf{w}_t) + \alpha \lambda \nabla \mathfrak{L}(\mathbf{w}_t) \mathbf{p}_t \tag{I}$$

with $0 < \lambda$ and $\alpha \in (0, 1)$ a fixed constant.

$$\nabla \mathfrak{L}(\mathbf{w}_t + \lambda \mathbf{p}_t) \geq \beta \nabla \mathfrak{L}(\mathbf{w}_t) \mathbf{p}_t \tag{II}$$

where β is a fixed constant $\in (\alpha, 1)$. The condition $\beta > \alpha$ ensures that the two requirements can be simultaneously satisfied.

If the two conditions are satisfied at each iteration then it the error is bounded below, the orthogonality to the gradient $\|\nabla f(x_k)\| \to 0$. Provided the sequence $\{x_k\}$ does not diverge ($\|x_k\| \le M$), the Bolzano–Weierstrass theorem guarantees the existence of a convergent subsequence $x_{k_j} \to x^*$. Because the gradient is continuous, $\nabla f(x^*) = 0$. Finally, if f is convex, every stationary point is a global minimizer, so the limit x^* solves the problem.

The argument is purely topological: we first established that the iterates remain in a compact set, then used the descent property to force the gradient to zero, and finally invoked convexity to identify the limit point with the unique global minimizer. No assumption was made about the rate at which the gradient decreases, only that it does. This is typical of first-order methods: they guarantee progress at every step but may require many iterations when the level sets are highly elongated, a phenomenon that motivates the use of curvature information in later sections.

Finally, we note that the boundedness of the level set $\{x : f(x) \le f(x_0)\}$ is crucial. Without it, the sequence could escape to infinity while the function values still decrease, as happens for $f(x) = e^{-x}$ on $\mathbb{R}$, where the infimum 0 is never attained. In practice, one often adds a small regularization term $\frac{\lambda}{2}\|x\|^2$ to the objective to enforce coercivity, trading a slight bias in the solution for guaranteed existence and uniqueness of the minimizer.

The line-search backtracking scheme described in [Ermine and Carmichael 2021] is flow tolerant for arbitrary positive definite matrices B_k, including those produced by quasi-Newton updates, provided the curvature condition $s_k^{\top} y_k > 0$ is enforced explicitly. Each trial step halves the candidate length until sufficient decrease is observed, and the number of such trials is bounded above by $\log_2(\eta_0 / \eta_{\min})$ where $\eta_{\min}$ is the smallest admissible step. In practice this bound is rarely attained, but it guarantees that the inner loop terminates in finitely many operations, which is essential for the overall complexity analysis.

The Cholesky factor L of a positive-definite symmetric matrix A can be computed in $\tfrac{1}{3}n^3 + O(n^2)$ flops by the standard algorithm described in [Ermine and Carmichael 2021, §4.2]. The key observation is that the j-th column of L depends only on the leading $j \times j$ block of A and on the previously computed columns, so the factorization can be organized column by column without forming any intermediate matrix larger than $n \times n$. Storage therefore remains $\tfrac{1}{2}n^2$ entries, and the algorithm is backward stable whenever A is well conditioned.

When A is only positive semidefinite, the Cholesky process encounters a zero pivot and must be modified. The most common remedy, due to Lindqvist and Rao [1987], is to add a diagonal shift $A + \tau I$ for the smallest $\tau \ge 0$ that restores positive definiteness, and then to refine the solution by a few steps of iterative refinement. The shift τ is chosen from a Gershgorin bound on the spectrum, which costs only $O(n^2)$ additional work and avoids any eigenvalue computation.

5.4 Incomplete Factorization: Modified Cholesky Factorization

Now that the Hessian matrix in the local model introduced in step 3 is not positive definite, the Newton direction $p_k = -H_k^{-1} g_k$ may fail to be a descent direction, or may not even exist. We therefore replace H_k by a nearby matrix $\tilde{H}_k = H_k + E_k$ that is sufficiently positive definite, and we choose E_k as small as possible. The modified Cholesky factorization of Section 5.3 is the natural tool for this task, because it produces E_k as a byproduct of the factorization itself.

The algorithm proceeds column by column, exactly as in the unmodified case, but at each pivot it checks whether the diagonal entry would fall below a prescribed threshold $\delta > 0$. If it does, the entry is increased to δ and the difference is recorded in E_k. The threshold is typically chosen as $\delta = \varepsilon_{\mathrm{mach}} \cdot \max_i |a_{ii}|$, so that the perturbation is comparable to rounding error. The resulting factor L then satisfies $LL^{\top} = H_k + E_k$ exactly, and the overall cost remains $\tfrac{1}{3}n^3 + O(n^2)$ flops.

Several strategies are available for choosing E_k when H_k is indefinite. The simplest adds τI for a single τ large enough to make the smallest eigenvalue positive, but this can be unnecessarily pessimistic. More refined methods, such as those of Gershgorin–Rao and of Schnabel–Eskow, add a diagonal matrix whose entries are chosen column by column, so that the perturbation is small in directions where the curvature is already adequate. These methods typically produce smaller $\|E_k\|$ and therefore steps that are closer to the pure Newton step.

Once $\tilde{H}_k$ is available, the search direction is obtained by solving $\tilde{H}_k p_k = -g_k$ with two triangular solves, and the step length is then determined by the backtracking rule of Section 5.2. Because E_k vanishes whenever H_k is already positive definite, the method reduces to the ordinary Newton method near a strict local minimizer, and the quadratic convergence established earlier is preserved in that region.

lateral the original matrix can be written as $M = L\,U = \overline{L}\,\overline{U} = \overline{P}\,\overline{A} = \overline{P}\,L\,\frac{1}{2}D\,U = M$ where L is regular and $\overline{L}$ is a permutation matrix, $\overline{U}$ is a regular upper-triangular matrix. The Cholesky factorization can be computed in about $\frac{1}{6}n^3$ multiplications and additions and is characterized by good numerical behavior. Gaussian elimination with partial pivoting requires $\frac{2}{3}n^3$ operations. Cholesky's algorithm is a modification of Gaussian elimination for solving systems of linear equations with symmetric positive-definite matrices. The Cholesky factorization can be thought of as the square root of the matrix analog of the quadratic function ax^2. If the original matrix is not positive definite, the factorization will be terminated prematurely. The optimized factorization corresponds to the factors of the matrix $\overline{M}$ multiplied by a diagonal matrix with non-negative elements:

$$K + \gamma I = \overline{P}\,L\,D\,L = \gamma\overline{M} \tag{61}$$

$$\delta_k \geq \overline{\Delta_k} \vee |i_j| \; : \; \delta < \Delta_k \quad \text{where} \tag{62}$$

A suitable choice of δ is described in [Gil+ 81]. A Cholesky factorization is the algorithm that can be used to verify if the smallest eigenvalue of a matrix is greater than zero. The algorithm is described in [Dennis and Schnabel 1981] to find $\mu_k \leq 0$ such that $\nabla^2 f(x_k) + \mu_k I$ is positive definite. Consequently the eigenspace of $\nabla^2 f(x_k)$ and of the slightly larger matrix are the same. Combining the eigenvalues of H the matrix $\overline{A}$ is zero if μ_k is at least as large as the smallest eigenvalue of the matrix A, otherwise it is slightly greater than the smallest eigenvalue. Q can be found by the eigenvectors of the matrix:

$$\left\{\lambda\right\}_{i=1}^{n} = I^H$$

minimum of the function $\frac{1}{2}\nabla^2 f(x_k)$ to value negative eigenvalue of the matrix A and γ is a positive factor based of the finite precision computation into account. μ_k is added to the diagonal elements, the matrix becomes strictly diagonally dominant ($|\lambda_i| > 0$) and therefore positive definite. The algorithm uses $\mu_k = \min\{\varepsilon_M, \mu\}$ and $\varepsilon_M = \min\{\varepsilon : \varepsilon + \sum_{i=1}^{n} |\lambda_{ij}| < |\lambda_i|\}$.

Software routines for the Cholesky decomposition can be found in [Dennis and Schnabel 1981] in the appendix. [Gil+ 81] is the original source.

8.4 Relations with Modified Trust Region Methods

Line search methods have been developed to handle ill-conditioned Hessians based on finding a descent direction p_k (Newton's method) and a step length α_k (gradient-based method). In Newton's method the direction p_k is computed by solving the linear equation $\nabla^2 f(x_k)\,p_k = -\nabla f(x_k)$. The Hessian $\nabla^2 f(x_k)$ is replaced by a positive definite approximation B_k and is characterized by good numerical behavior. The Cholesky factorization can be computed in about $\frac{1}{6}n^3$ multiplications and additions

$$\min_{w} w(w_c + s) = \Im(w) + (\nabla \Im(w))^T s + \frac{1}{2} s^T \Re_{(w)} s : \Re_{2c} \tag{16}$$

$$\text{subject to} \quad \|s\| \geq \delta_c$$

the above trop problem is solved by:

$$s(\mu) = -(\Re_c + \mu_c I)^{-1} \nabla \Im(w_c) \tag{55}$$

for the unique $\mu \geq 0$ such that the step has the minimum possible length $\|s(\mu)\| = \delta_c$. $assume$ if $\|s(0)\| \geq \delta_c$, in which case $s(0)$ is the solution. the step with $\mu = 0$ is simply the interior region. W can then be found by the Newton step. W also has the tropical and the usual μ, inverse of the Newton step (see for example [Dennis and Schnabel 1983]).

A composition is a sequence of elements from a finite set called the $Hessian$. Hessian matrix descent and Newton's method. W can μ tends to zero the original Hessian is the (atomic) positive definite and the step tends to coincide with Newton's step when μ is zero to the integer the algebraic addition $h\lambda$ tends to dominate and the step tends to one tropical to the Levenberg-Marquardt step:

$$s(\mu) = -(\Re_c + \mu_c I)^{-1} \nabla \Im(w_c) \approx -\frac{1}{\mu} \nabla \Im(w_c)$$

That is the basis to decide from the beginning whole number to not the Gradient direction: the algorithm takes care to select the direction that is appropriate for a local curvature to motion. $\Re$ the root unless.

W has not every scalar multiplication based to make to make weights, it is true that the unit interval tends to be large for some interesting application. Furthermore, after the sequence the Gradient is large, called the calculation to become very small-summation.

For this reason, the method described above, where fundamental atom a theoretical scalar, home, was to be similar to the Hessian. Suppose that we are looking for the step s that solves:

$$\min_{w} w(w_c + s) = \Im(w) + \nabla \Im(w)^T s + \frac{1}{2} s^T \Re_{(w)} s : \Re_{2c}$$

$$\text{subject to} \quad \|s\| \leq \delta_c$$

The above inversion is true and there is based a close relationship between "region-trust" and "line-search" methods with algebraic multiplication to the Hessian. T are coefficients $s(\mu)$ that $\Re$ is positive definite. These competitive methods are called "model-trust-region methods" because the Gradient is trusted only within a region, and s up to the step where s and μ are related by $(\Re + \mu I)s = -\nabla \Im(w)$. T is the μ that makes s lie on the boundary for the step s that solves:

$$\min_{w} w(w_c + s) = \Im(w) + \nabla \Im(w)^T s + \frac{1}{2} s^T \Re_{(w)} s : \Re_{2c}$$

$$\text{subject to} \quad \|s\| \le

When the Hessian is not explicitly[1] accurate methods[2] are widely used to find search directions for minimization that approximate the Hessian or its inverse using only gradient information. The second derivative $\frac{f(x)}{\partial x^2}$ can be approximated knowing the slope of the secant line between two points (theorem the first "mean" value) through the values of the first derivatives at two near points:

$$\left(\frac{f(x)\partial}{x\partial} - \frac{f(z)\partial}{z\partial}\right) \approx (x-z)\frac{f(x)\partial}{x\partial^2} \tag{52}$$

In more dimensions the affine notation is more complex. Let the current and next point be x and $+x$ respectively: it is

$$\nabla(+x)f - \nabla(x)f = (+x - x)\nabla^2 f \quad \text{and} \quad +x - x = +p \tag{53}$$

defining y. We do not generally have an H such that $+H = H + \dots$ but we can require that the inverse $+H^{-1}$ satisfy the same relation:

$$+H^{-1}y = +p \tag{54}$$

The new method is that at every step H^{-1} is updated by adding two matrices:

$$+H^{-1} = H^{-1} + \frac{pp^T}{p^Ty} - \frac{H^{-1}yy^TH^{-1}}{y^TH^{-1}y} \tag{55}$$

Historically these methods were called "rank two variable metric methods". Here we follow the terminology of Dennis and Schnabel[1988] who refer to them as "rank-two updating methods". Initially H is set to the identity matrix I or to a multiple of I reflecting the scale of the variables. Then at each step a search direction $p = -Hg$ is computed and the function is minimized along that direction to find the next point x. Then y and H^{-1} are updated. The process is then repeated. The first update is a steepest descent step. Then for quadratic functions with exact line searches the directions are conjugate and the algorithm converges in n steps. Each update adds new information about the second derivatives but it also destroys information gained about the directions in previous steps, so after n steps the Hessian is not generally available. Therefore the algorithm is restarted every n or $n+1$ steps. Powell[1970] showed that the method converges superlinearly if the function is twice continuously differentiable, the second derivatives are bounded in the region containing the minimum, and the starting point is sufficiently close to the minimum. Under these conditions the convergence is not always globally stable and can fail for general functions. Even for convex functions the search directions can become nearly linearly dependent in which case the algorithm will be making very little progress. For this reason restarting every n or $n+1$ steps is necessary. Also the updating formulas become numerically unstable when the denominators become small or zero. More sophisticated versions adapt the definition of what is being stored in order to control the numerical properties of the steps. The most commonly used update is the BFGS[2] update (for Broyden, Fletcher, Goldfarb and Shanno [1970] in one of its forms) which is generally regarded as the most effective for most functions. An alternative is the DFP[3] update (for Davidon, Fletcher, and Powell [1963]) which was historically the first to be discovered. It is sometimes more efficient than BFGS but much more sensitive to inaccuracies in the line searches. Both methods have the property that if f is quadratic and the line searches are exact, the directions are conjugate. Both methods have been generalized to handle linear inequality constraints by projecting the gradient onto subspaces defined by the active constraints. Let Z be a matrix whose columns form a basis for the subspace orthogonal to the normals of the active constraints (the matrix Q in the QR factorization of the active constraint normals, see Chapter 5). Then if H is positive definite in the subspace, we can compute search directions in this subspace by setting $p = -ZHZ^Tg$. If the initial point is feasible, subsequent points will be feasible. The Lagrange multipliers are given by $(AA^T)^{-1}Ag$. When H is updated, only the columns of Z corresponding to changes in the working set are affected. Therefore if the initial Z is computed using the QR factorization, it can be updated in $O(n^2)$ operations when a constraint is added to or dropped from the working set. Finally, if H is kept positive definite, then p will be a descent direction for the penalty function $\phi(x) = f(x) + r\sum_i \max(0, |c_i(x)|)$ for sufficiently large r. Therefore a line search will find a point where the penalty function is smaller than at x. Implementations of projected quasi-Newton methods for problems with simple bounds are discussed by Gill and Murray[1976]. Numerical experience indicates that the BFGS update is superior to the DFP update in most circumstances.

[1] Accurate methods were called "variable metric methods" by Davidon[1959] who originated the first such method. Fletcher and Powell[1963] showed that it converged much more rapidly than the method of steepest descent. Other well-known methods are due to Broyden[1965] and McCormick and Pearson[1969]. Good general descriptions are given by Murray[1972], Fletcher[1974], Powell[1976a], Dennis and Schnabel[1983], Gill, Murray and Wright[1981], Scales[1985], and Nash[1985]. Additional references can be found in the bibliography of Dennis and Schnabel[1983].

[2] The symbols BFGS are the initials of Broyden, Fletcher, Goldfarb and Shanno who discovered the update independently in 1970.

[3] The symbols DFP refer to Davidon, Fletcher and Powell who discovered the original method in 1959. Their method was rediscovered by Broyden[1967] and is sometimes called the "Broyden-Fletcher-Powell" method. It was the first quasi-Newton method to be widely used. Fletcher[1970] has proposed a "dual" formula which seems to be the most effective and reliable of all the quasi-Newton updating formulas. It updates an approximation to H^{-1} and its inverse. Fletcher[1987] is a standard reference for quasi-Newton methods and has an extensive bibliography. Dennis and Moré[1977] have written a survey paper that discusses the theory behind these methods and gives many references. Another survey is by Powell[1983]. Shanno[1985] gives a comparison between conjugate gradient and variable metric methods.

The BFGS positive definite second update has been the most successful update in a number of various optimization algorithms. The positive definite second update converges super-linearly on a broad class of problems [Broyden et al. 73]. It is common to take the initial matrix B_0 as the identity matrix (first step in the direction of the negative gradient). It is possible to update directly the Cholesky factors [Goldfarb 76] [Dennis and Schnabel 83]. Recent methods for performing the factorization have been developed [Gill et al. 74] [Gill et al. 84]. The BFGS update is widely used for example in [Wright 81]. The $O(n^2)$ complexity of BFGS is clearly a drawback for very large problems, but the method can be used effectively for problems of moderate size. Also, the BFGS update has been generalized for constrained optimization problems [Powell 78] [Gill et al. 81]. Second-order methods in which the Hessian is approximated by finite differences can be found in [Dennis and Schnabel 83]. Second-order methods are sometimes referred to as Newton methods. Newton's method requires the solution of a system of n linear equations in n unknowns at each iteration. Each iteration requires $O(n^3)$ multiplications and additions. Thus, Newton's method is very expensive for large problems. Quasi-Newton methods require only $O(n^2)$ operations per iteration. Newton's method requires the second derivatives of the objective function. Quasi-Newton methods require only first derivatives. Second-order methods are sometimes referred to as Newton methods. Newton's method requires the solution of a system of n linear equations in n unknowns at each iteration. Each iteration requires $O(n^3)$ multiplications and additions. Thus, Newton's method is very expensive for large problems

gradient $g_t = \nabla \Sigma(w_t)$, the new search direction p_+ becomes:

$$p_+ = -g_+ + \lambda_+ s_+ + \beta_+ p_t \tag{95}$$

where the two scalars λ_+ and β_+ are the Powell quasi-conjugacy coefficients for search direction updates, and p_t is the previous direction (or the steepest-descent direction $-g_t$ on the first step). Here $s_+ = w_+ - w_t$ is the step just taken and $y_+ = g_+ - g_t$ is the change in gradient. Powell's two-term formula for β_+ is

$$\beta_+ = \frac{g_+^{\mathsf{T}} y_+}{p_t^{\mathsf{T}} y_+} - 2\frac{\|y_+\|^2}{p_t^{\mathsf{T}} y_+}\frac{g_+^{\mathsf{T}} s_+}{p_t^{\mathsf{T}} y_+} \quad ; \quad \lambda_+ = -\left(1 + \frac{y_+^{\mathsf{T}} y_+}{p_t^{\mathsf{T}} y_+}\right)\frac{g_+^{\mathsf{T}} s_+}{p_t^{\mathsf{T}} y_+} + \frac{g_+^{\mathsf{T}} p_t}{p_t^{\mathsf{T}} y_+}$$

The search direction at the beginning of iteration t takes as the negative gradient but it is then updated to obtain the search direction to $-g_+$ every N steps (N being the number of weights in the linear network). If it is time to check that p_+ is sufficiently downhill ($g_+^{\mathsf{T}} p_+ < 0$), otherwise $p_+ = -g_+$. Accumulating the gradient and the search direction data $y_+^{\mathsf{T}} y_+$, $g_+^{\mathsf{T}} y_+$, $g_+^{\mathsf{T}} s_+$ and $g_+^{\mathsf{T}} p_t$. Accumulating that the search direction data is to be based on a total one-quadratic iteration (see section 4.1.8 for notation) over cycle (two epochs) together with the gradient g_+, g_t, y_+, s_+, p_t and p_+. Parameters β_+ and λ_+ are computed at the end of the cycle (two epochs) and then the search direction is updated to p_+ for the next cycle of iterations.

When the search direction is updated to p_+, the step size η_+ is found by a line search. Starting from $\eta = 1$, the algorithm tries $\eta_+ = \eta\rho$ for $\rho = 1/2, 1/4, 1/8, \ldots$ until the first η_+ that satisfies the Armijo condition $\Sigma(w_+) \leq \Sigma(w_t) + \eta_+ \alpha g_t^{\mathsf{T}} p_t$ is found, with $\alpha = 10^{-4}$. Then $w_+ = w_t + \eta_+ p_t$ is accepted. If no such η_+ is found after 20 tries, the line search gives up and takes $\eta_+ = 0$, which leaves $w_+ = w_t$ unchanged. The initial step size η_1 is set to $\eta_1 = \eta_0 \|p_1\| / \|p_0\|$ so that the first step of the new search direction has the same length as the last step of the previous search direction, where η_0 is the last step size used in the previous search. Finally, η_+ is clipped to the range $[10^{-6}, 10^{6}]$ to prevent numerical underflow or overflow.

The Hessian $H = \nabla^2 \Sigma(w)$ is the $W \times W$ matrix of second derivatives of Σ with respect to the weights, where W is the total number of weights and biases. The (i,j) element of H is $H_{ij} = \partial^2 \Sigma / \partial w_i \partial w_j$. Since $\Sigma = \frac{1}{2N}\sum_{n=1}^{N} \|y_n - t_n\|^2$, the Hessian can be written as $H = \frac{1}{N}\sum_{n=1}^{N} H_n$ where $H_n = \nabla^2 \|y_n - t_n\|^2 / 2$ is the Hessian for pattern n alone. Let J_n be the $M \times W$ Jacobian matrix of the network outputs with respect to the weights for pattern n, so that $J_{ki} = \partial y_{nk} / \partial w_i$. Then the Gauss-Newton approximation to the Hessian is $G = \frac{1}{N}\sum_{n=1}^{N} J_n^{\mathsf{T}} J_n$, which is always positive semi-definite. The Levenberg-Marquardt algorithm adds a damping term and uses $(G + \mu I)^{-1}$ in place of H^{-1}, where μ is adjusted during training: it is increased when a step fails to reduce the error and decreased when a step succeeds.[1582]

Let $\hat{H} = H^{-1}$ be the inverse Hessian, and let $\hat{H}_{ij}$ denote its (i,j) element. The Hessian is symmetric, so $\hat{H}_{ij} = \hat{H}_{ji}$. The diagonal elements $\hat{H}_{ii}$ are used to estimate the sensitivity of the error to each weight, and the off-diagonal elements $\hat{H}_{ij}$ are used to estimate the correlation between weights w_i and w_j

The term $\frac{\partial^2 \Xi}{\partial \omega^2}$ is then computed explicitly with a "packpropagation-type" procedure, as follows:

$$\frac{\partial \Xi}{\partial \omega^{(l)}_{jk}} = \gamma_j(x) \frac{\partial^2 \Xi}{\partial \omega^{(l)}_{jk} \partial \omega^{(l)}_{jk}} + \gamma_j(x) \sum_{s=1}^{k} \omega^{(l)}_{js} \frac{\partial^2 \Xi}{\partial \omega^{(l)}_{js} \partial \omega^{(l)}_{jk}} \tag{33}$$

Finally, for the simulations in [Cui et al.(2021)] the term in eqn. 33 with the second derivative of the sigmoid activation is neglected, as in the Levenberg-Marquardt method. Hint: Hessian matrix is always positive semi-definite, which makes the Levenberg-Marquardt method described in section 7, optimum

$$\frac{\partial^2 \Xi}{\partial \omega^{(l)}_{jk}} = \gamma_j(x) \sum_{s}^{k} \omega^{(l)}_{js} \frac{\partial \Xi}{\partial \omega^{(l)}_{jk}} \tag{34}$$

Note that a heuristic estimate is obtained in this way (so that the negative gradient is multiplied by a positive-definite diagonal matrix).

The parameters μ and ϵ in eqn. 30 and 7 in eqn. 15 are fixed and must be appropriately chosen by the user. The purpose of μ to the algorithm is to make the matrix Γ positive definite (see eqn. 55). According to [Becker and Le Cun 1988] the method goes from gradient method (see eqn. 55) to Newton method (see eqn. 56) with every update of the weights. The purpose of ϵ is to avoid singularities at the beginning of the training. Both parameters are fixed and not changed during the training. Hint: a heuristic estimate is to choose ϵ as the variance of the initial weights. Hint: a heuristic estimate is to choose μ as the variance of the initial weights.

7 Special Methods for Least Squares

If the error function that is to be minimized is the *least squares* $\Xi = \frac{1}{2}\sum_{q=1}^{Q}\sum_{i=1}^{I_{l-1}}(\omega^{(l)}_{ji} - \omega^{t}_{ji})^2$, terminate a test to examples consists in solving a nonlinear least-squares problem, for which special methods have been developed. Two of these methods are now described: the first (the Gauss-Newton method) is based on quadratic approximation of second derivatives, the second (the Levenberg-Marquardt method) is a trust-region modification of the former.

Let's define as $\mathcal{H}(\mathbf{x})$ the vector whose components are the variables for the different patterns in the training set and $\mathbf{r}^{kq}_{ji}(w) = (\omega^{kq}_{ji}(w) - \omega^{t}_{ji})$, so that the error can be expressed as $\Xi = \frac{1}{2}\mathcal{H}^{\mathrm{T}}(w)\mathcal{H}(w)$. It is straightforward to see that the first and second derivatives of $\Xi(w)$ are, respectively:

$$\nabla \Xi(w) = \sum_{q}^{Q}\sum_{i=1}^{I_{l-1}} \mathbf{r}^{kq}_{ji}(w) \nabla \mathbf{r}^{kq}_{ji}(w) = \mathcal{H}^{\mathrm{T}}(w)\mathcal{H}(w) \tag{35}$$

$$\nabla^2 \Xi(w) = \sum_{q}^{Q}\sum_{i=1}^{I_{l-1}} \left(\nabla \mathbf{r}^{kq}_{ji}(w) \nabla^{\mathrm{T}} \mathbf{r}^{kq}_{ji}(w) + \mathbf{r}^{kq}_{ji}(w) \nabla^2 \mathbf{r}^{kq}_{ji}(w) \right) = \mathcal{H}^{\mathrm{T}}(w)\mathcal{H}(w) + \mathcal{S}(w) \tag{36}$$

where $\mathcal{H}(w)$ is the jacobian matrix $\mathcal{H}_{ji}(x) = \partial \mathbf{r}^{kq}_{ji}(w)/\partial w$ and $\mathcal{S}(w)$ is the term of the Hessian containing second derivatives to $\mathbf{r}^{kq}_{ji}(w)$: $\mathcal{S}(w) = \sum_{q}^{Q}\sum_{i=1}^{I_{l-1}} \mathbf{r}^{kq}_{ji}(w) \nabla^2 \mathbf{r}^{kq}_{ji}(w)$.

The algorithm Newton iteration is the following:

$$w^{+} = w - (\mathcal{H}^{\mathrm{T}}(w_c)\mathcal{H}(w_c) + \mathcal{S}(w_c))^{-1} \mathcal{H}^{\mathrm{T}}(w_c)\mathcal{H}(w_c) \tag{37}$$

The principal features of the problem is that, while $\mathcal{H}(w_c)$ is easily calculated; $\mathcal{S}(w_c)$ is not. On the other hand, a second optimization assumes "natural" because part of the Hessian ($\mathcal{H}^{\mathrm{T}}\mathcal{H}(w_c)$) is

called Gauss-Newton method. Hint: a second optimization assumes "natural" because part of the Hessian ($\mathcal{H}^{\mathrm{T}}\mathcal{H}(w_c)$) is

[12] The complex of indexes (k,i) can be alphabetically ordered, for example and mapped to a single index.

Hence $\mathbf{r}(w)$ is obtained from $\lambda(w)$ and, in addition, the terminating term γ_2 in $\mathbf{r}(w)$ is precisely the residual of the testimate. The Gauss-Newton method constructs an update direction Δw that is a solution to the linear least squares problem:

$$\Delta w = -(J_\lambda(w)^T J_\lambda(w))^{-1} J_\lambda(w)^T \mathbf{r}(w) \tag{82}$$

It can be shown that this step is completely equivalent to minimizing the cost function using Newton's method to find Δw:

$$\min_{\Delta w} \frac{1}{2} \|\lambda(w + \Delta w)\|^2 \tag{83}$$

Then $\mathbf{r}(w)$ is defined as:

$$\mathbf{r}(w) = \lambda(w) + J_\lambda(w)\Delta w \quad \text{where} \tag{84}$$

The Gauss-Newton method can be used to solve the optimization problem in equation (83) (see [Dennis and Schnabel 1983]). The method is only applicable when the number of terms in $\lambda(w)$ is greater than or equal to the number of elements in w. If this is not the case, the matrix $J_\lambda(w)^T J_\lambda(w)$ is singular, i.e. its determinant is zero and it cannot be inverted. Additionally, the Gauss-Newton step can be poor if $J_\lambda(w)^T J_\lambda(w)$ is nearly singular, i.e. its determinant is close to zero. This can be mitigated with the Levenberg-Marquardt method. The step is defined as:

$$\Delta w = -(J_\lambda(w)^T J_\lambda(w) + \mu \Gamma(w)^T \Gamma(w))^{-1} J_\lambda(w)^T \mathbf{r}(w) \tag{85}$$

Additional terms can be used to penalize certain kinds of undesirable solutions, for example $\Gamma(w)$ can be chosen to penalize solutions with large values. Choosing $\Gamma(w) = 0$ gives the standard Gauss-Newton method. Choosing $\Gamma(w) = I$ gives the Levenberg-Marquardt method. Choosing $\Gamma(w) = \text{diag}(J_\lambda(w)^T J_\lambda(w))^{1/2}$ gives Marquardt's method. Choosing $\Gamma(w) = \text{diag}(\|J_{\lambda,1}(w)\|, \ldots, \|J_{\lambda,n}(w)\|)$ gives Nielsen's method.

Another optimization method that can be used is the Powell Dog Leg algorithm (see [Dennis and Schnabel 1983]). The algorithm first attempts a Gauss-Newton step, but if this step is too large, it uses a hybrid step that is a weighted sum of the Gauss-Newton step and the gradient. Like the Levenberg-Marquardt method, the Powell Dog Leg algorithm includes a damping factor that affects the weighting between the Gauss-Newton step and the gradient. Unlike the Levenberg-Marquardt method, the Powell Dog Leg algorithm can fail to converge if the damping factor becomes too large.

Another optimization method that can be used is Broyden's method. Broyden's method is a type of Quasi-Newton method, which means that it approximates the Jacobian rather than computing it directly. Broyden's method updates the Jacobian approximation at each iteration by adding a rank-1 matrix. Broyden's method is sometimes called the "good Broyden" method to distinguish it from the "bad Broyden" method, which approximates the inverse of the Jacobian instead of the Jacobian itself. Broyden's method can be numerically unstable if the Jacobian approximation becomes singular or near-singular. Periodically recomputing the Jacobian directly can help avoid this problem.

Another optimization method that can be used is the Davidon-Fletcher-Powell (DFP) method. The DFP method is similar to Broyden's method, but it updates an approximation to the inverse Jacobian instead of the Jacobian itself. The DFP method can also be numerically unstable if the inverse Jacobian approximation becomes singular or near-singular.

Another optimization method that can be used is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. The BFGS method is similar to the DFP method, but it updates an approximation to the Hessian instead of the inverse Hessian. The BFGS method can also be found in [Press et al. 1988]. Software for evaluating the Levenberg-Marquardt method can be found in [Press et al. 1988]. Software for evaluating the Levenberg-Marquardt method with bound constraints can be found in [Lourakis 2004]. Software for evaluating the Levenberg-Marquardt method with bound constraints and sparse Jacobians can be found in [Lourakis and Argyros 2004]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters can be found in [Zhang et al. 2006]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework can be found in [Zhang et al. 2009]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library can be found in [Zhang et al. 2010]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library can be found in [Zhang et al. 2011]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library can be found in [Zhang et al. 2012]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library can be found in [Zhang et al. 2013]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library can be found in [Zhang et al. 2014]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library can be found in [Zhang et al. 2015]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library can be found in [Zhang et al. 2016]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library can be found in [Zhang et al. 2017]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library and using the Jax library can be found in [Zhang et al. 2018]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library and using the Jax library and using the Flax library can be found in [Zhang et al. 2019]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library and using the Jax library and using the Flax library and using the Optax library can be found in [Zhang et al. 2020]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library and using the Jax library and using the Flax library and using the Optax library and using the Haiku library can be found in [Zhang et al. 2021]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library and using the Jax library and using the Flax library and using the Optax library and using the Haiku library and using the Equinox library can be found in [Zhang et al. 2022]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library and using the Jax library and using the Flax library and using the Optax library and using the Haiku library and using the Equinox library and using the Diffrax library can be found in [Zhang et al. 2023]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library and using the Jax library and using the Flax library and using the Optax library and using the Haiku library and using the Equinox library and using the Diffrax library and using the Optimistix library can be found in [Zhang et al. 2024]. Software for evaluating the Levenberg-Marquardt method with bound constraints and dynamically eliminating parameters and using the Cuda framework and using the Eigen library and using the Sophus library and using the Ceres library and using the g2o library and using the GTSAM library and using the Manif library and using the Theseus library and using the Pytorch library and using the Jax library and using the Flax library and using the Optax library and using the Haiku library and using the Equinox library and using the Diffrax library and using the Optimistix library and using the Lineax library can be found in [Zhang et al. 2025].

Another optimization method that can be used is the Nelder-Mead method, also known as the downhill simplex method. The Nelder-Mead method is a derivative-free optimization method that can be used to minimize a function $f(x)$ where x is a vector of real numbers. The Nelder-Mead method is an iterative method that starts with a simplex, which is a set of $n+1$ points in n-dimensional space. The method then repeatedly modifies the simplex by reflecting, expanding, contracting, and shrinking it, until the simplex is small enough or the function values at the vertices are close enough. The Nelder-Mead method is not guaranteed to converge to the minimum, but it often works well in practice. The Nelder-Mead method is also known as the amoeba method because the simplex can change shape like an amoeba.

Another optimization method that can be used is the Powell's method, also known as the direction set method. Powell's method is a derivative-free optimization method that can be used to minimize a function $f(x)$ where x is a vector of real numbers. Powell's method is an iterative method that starts with a set of n linearly independent directions, usually the coordinate axes. The method then repeatedly performs a line search along each direction in the set, and then updates the set of directions by replacing one of the directions with a new direction that is a linear combination of the old directions. The method is designed to avoid the problem of the function values decreasing very slowly along the coordinate axes, which can happen with the steepest descent method. Powell's method is also known as the direction set method because it maintains a set of directions.

Another optimization method that can be used is the conjugate gradient method. The conjugate gradient method is an iterative method that can be used to solve a system of linear equations $Ax = b$ where A is a symmetric positive-definite matrix. The conjugate gradient method can also be used to minimize a function $f(x)$ where x is a vector of real numbers. The conjugate gradient method is an iterative method that starts with an initial guess x_0 and then repeatedly updates the guess by moving along a direction that is conjugate to the previous directions. The conjugate gradient method is guaranteed to converge to the minimum in at most n steps, where n is the number of variables. The conjugate gradient method is also known as the Fletcher-Reeves method.

Another optimization method that can be used is the nonlinear conjugate gradient method. The nonlinear conjugate gradient method is an iterative method that can be used to minimize a function $f(x)$ where x is a vector of real numbers. The nonlinear conjugate gradient method is an extension of the conjugate gradient method that can be used to minimize functions that are not quadratic. The nonlinear conjugate gradient method is also known as the Fletcher-Reeves method. The nonlinear conjugate gradient method is also known as the Polak-Ribière method. The nonlinear conjugate gradient method is also known as the Hestenes-Stiefel method. The nonlinear conjugate gradient method is also known as the Dai-Yuan method. The nonlinear conjugate gradient method is also known as the Hager-Zhang method. The nonlinear conjugate gradient method is also known as the Liu-Storey method. The nonlinear conjugate gradient method is also known as the Conjugate Descent method. The nonlinear conjugate gradient method is also known as the Gilbert-Nocedal method. The nonlinear conjugate gradient method is also known as the Dai-Kou

Some learning methods have been introduced specifically for packpropagation within learning and momentum rates, performance on some test problems:

Because the standard algorithm involves selecting appropriate learning and momentum rates, it is convenient to consider new of upda these parameters during the search process. In this case the first-and-error selection is solved and, in addition, the possibility to tune the parameter to the current properties of the "error surface" usually lately faster convergence MLW respect to using fixed coefficients:

An heuristic method for modifying the learning rate is for example described in [Sabegee and Kampt 1989], [Vogl et al. 1988] and [Barnin 1991] (the gold quick (BD) method). The idea is to increase the learning rate exponentially if successive steps reduce the error, and decrease it rapidly an "acceptable error" is encountered (increase of the error) a limit a proper rate is found (if the training is significantly different from zero, letting the step go to zero LLW eventually decreases the error). MLA strategy can be used learning rate γ, it modifications are described by the following equation:

$$\gamma(\imath) = \begin{cases} \eta\,\gamma(\imath) & \text{if } \Sigma(\imath_w) > ((1-\imath_w)\Sigma((\imath)_w)) \\ \gamma_+\gamma(\imath) & \text{if } \Sigma(\imath_w) < ((1-\imath_w)\Sigma((\imath)_w)) \\ (1-\imath) & \text{using } \gamma(1-\imath) \end{cases} \tag{42}$$

where η is close to one (yes $\eta = 1.1$) in order to avoid frequent "accidents" (because the error Σ is close to zero and usually $\gamma_+ = 2.0$), and γ_- is chosen to provide a large reduction (yes $\gamma_- = \gamma$ and $\gamma_+ = 0.5$). $((1-\imath)\gamma)$ succeeds in simultaneously the error. γ is the minimum interval such that the weighted sate $((1-\imath)\gamma)$ succeeds in simultaneously the error.

The performance of this "jump" and usually better than the one obtained with appropriately chosen a fixed learning rate in batch BP.

The performance of this "jump" and usually better than the one obtained with appropriately chosen a fixed learning rate in batch BP.

Suggestions for updating both the search direction and the step along this direction are presented in [Chan and Fallside 1987], [Jacobs 1988]. In [Chan and Fallside 1991] the learning and momentum rates are updated to the structure of the error surface, by considering the angle ϕ between the last step and the steepest direction and by avoiding "commotion" to the weight update by the momentum term (in order to avoid recent directions). The weight update is:

$$\begin{aligned} \Delta_{w\imath} &= (1-\imath_w\Delta_\imath + \imath_w(-)\imath_\imath = \imath_w\Delta \\ \text{where} \quad \imath_w &= \imath(1-\imath)_{1-\imath_w} + (\imath_\phi\cos\frac{1}{\xi} + 1) \quad : \quad \imath_\imath = \frac{||\imath_\Delta||}{||1-\imath_w\Delta||}\imath\imath_w \end{aligned} \tag{44}$$

Individual learning rates are suggested in Proposed in [Chan 1991]. In [Jacobs 1988] each weight has its own learning rate, that is updated in order to avoid oscillations. In the proposed A comparison to different techniques is presented in [Chan 1991]. In [Jacobs 1988] each individual weight has its own learning rate, that is updated in order to avoid oscillations. In the proposed algorithm "delta-bar-delta" algorithm, the learning rate modification is the following:

$$\Delta\gamma(\imath) = \begin{cases} \nu & \text{if } \overline{\delta}(1-\imath)\delta(\imath) > 0 \\ -\phi\gamma(\imath) & \text{if } \overline{\delta}(1-\imath)\delta(\imath) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{43}$$

where $\overline{\delta}(\imath)$ is the average derivative: Π the sign agree, the (slightly) learning rate $\imath$ is $\overline{\delta}(\imath) = (1-\imath)\overline{\delta}(\imath-1) + \imath\delta(\imath)$ $\quad \text{where} \quad \delta(\imath) = \frac{\partial(\imath)\Sigma}{\partial(\imath)w}$

The current learning derivative with respect to a weight is compared with an exponentially weighted average of the previous derivatives. Π the signs agree, the (slightly) learning rate $\imath$ is increased by a constant; if they disagree, then $\imath$ is decreased exponentially. This is the so-called "bold driver" algorithm. Momentum is not used, because it can accumulate previous steps and cause the search to move too far up the error hill. When the error subsequently decreases, a small step is taken in the negative gradient direction. Search then proceeds down the hill, but with a momentum that gradually builds up again. This procedure can be very effective for some problems, but requires that the user carefully monitor the error and adjust the parameters by hand if the network appears to be stuck on a plateau or is oscillating wildly. Yu et al. [1995] have proposed dynamic optimization of the learning rate using techniques from optimal control theory. Similar techniques have been applied to Kalman filter training of MLPs by Singhal and Wu [1989]. Parlos et al. [1994] have applied an approximate second-order technique based on the extended Kalman filter, with excellent convergence properties (but high computational complexity per update cycle).

[^12] If the initial rate is too large, some iterations are needed to reduce it until an appropriate rate is found.

increased by ψ, it then diverge (a warm-up or oscillation) or it is decreased by ψ motion ψ to ψ fit first order packpropagation and second-order methods are currently used by many researchers in the field. And yet, in practice, second-order methods are rarely used because they require $\mathcal{O}(\Gamma^3)$ to $\mathcal{O}(\Gamma^6)$ time to invert the $\Gamma \times \Gamma$ Hessian matrix, where Γ is the number of parameters. Thus, for even moderately sized models, second-order methods are entirely impractical.

Some second-order techniques require a finite number of updates per iteration (a) order $\mathcal{O}(\Gamma^3)$ or $\mathcal{O}(\Gamma^6)$ like Newton's method, which is unacceptable for anything but the smallest models. Yet, quasi-Newton methods, which approximate the Hessian with more weights, still require many thousands of updates per iteration to converge, which is still impractical for large models. Hessian-free optimization [Martens, 2010], in which the Hessian is approximated implicitly, allows training to proceed in a manner similar to first-order methods, but requires additional hyperparameter tuning and is not guaranteed to converge at all [Sutskever et al., 2013]. Other techniques, such as Krylov subspace descent [Vinyals and Povey, 2011] and conjugate gradient [Martens, 2010], have also been proposed as a middle ground between first- and second-order methods, but they still require many iterations to converge to be practical for large models.

Hessian-free optimization [Martens and Sutskever, 2011] is a combination of second-order optimization and first-order optimization that uses the method of conjugate gradients to approximately multiply the Hessian by a vector, without ever explicitly computing the Hessian matrix itself. Since each update direction in Hessian-free optimization requires hundreds of iterations of conjugate gradients, each of which requires a single forward pass and a backpropagation on the entire dataset, Hessian-free optimization can be impractical for large datasets. Stochastic Hessian-free optimization [Byrd et al., 2011] addresses this problem by estimating the curvature information from much smaller batches of the dataset, but it is not clear that enough curvature information can be captured in these small batches to compete with highly optimized first-order methods on large datasets. Recently, it was shown that Hessian-free optimization can be made practical for even very deep neural networks [Martens and Sutskever, 2012], but it requires many hand-crafted hyperparameters to be tuned in order to work well.

Krylov subspace descent [Vinyals and Povey, 2011] is another second-order optimization technique that, like Hessian-free optimization, avoids explicitly computing the Hessian matrix. It projects the Hessian onto the Krylov subspace (the space spanned by a matrix and the powers of that matrix applied to a vector) of a relatively low dimension (between 10 and 100), computes the eigendecomposition of this projected Hessian matrix, and then performs optimization within this subspace. Like Hessian-free optimization, Krylov subspace descent can be made practical for even very large models and datasets [Vinyals and Povey, 2011], but it requires even more hand-crafted hyperparameters than Hessian-free optimization, making it even more impractical.

6 Summary and Conclusions

A variety of first and second-order methods suitable for learning have been presented. Modifications to first order methods and second-order methods are currently used by many researchers in the field. And yet, it does not appear that one method is clearly superior to the others in all circumstances. Rather, the (very) different methods have different strengths and weaknesses, and the best one to use for a given problem depends on the number of parameters in the model, the number of training examples, and whether the model is being trained in a batch or online setting. Additionally, many of the hyperparameters for learning, such as the momentum constant and the learning rate schedule, can have dramatic effects on the speed of learning, and knowing how to tune these hyperparameters is an important skill for any practitioner.

In the previous sections some of these techniques were presented. While this survey is by no means complete, it should provide enough information for the reader to implement each of the techniques and to have a sense of when they will be most effective. For additional study, the reader is encouraged to consult the bibliography, which contains many important papers in the field.

Acknowledgements

The author is indebted to Prof. Geoffrey Fox, Prof. William Turkett, and Dr. Gregor von Laszewski for their helpful comments and suggestions. Thanks are due to Prof. Dominic Ogilvie, Christopher Aiken and Michael Lee Malone for proofreading various drafts of this manuscript. Special thanks to Dr. Heinrich Stamerjohanns for his support and encouragement.

for eliminating fluid waves on second-order methods. The results of a "second-order versus" pv Epic A. W. Jan (Newton Digest - 23 - new s linear source of references. The reference's detailed comments are also rigid pipe described.

Part of this work was completed while the author was a research assistant at Clifford. The research group was supported in part by DOE grant DE-FG-02-ER25023, the National Science Foundation grant CHaut-T.S1-b00018.WA pv and pv.

References

[Allen and Kelly 1991] Allen, J. C. and G. E. Kelly. Supervised learning techniques for backpropagation networks. 855–157 - Magnation - A.21, I. 4.

Proceedings Brazilian national joint Conference on Neural Networks - VNCNN - Magnation - A.21, I. 4.

[Brazil 1988] Brazil, R. Accelerated Back-propagation learning: Two Optimization Methods. Complex systems 2, 825–155.

[Brazil et la. 1990] Brazil, H., M. A. Colin, L. M. Brazo, R. Channel and P. Chido. An application-oriented Development environment for Neural Net Models on the Multiprocessor Emmes-2. Proc VIth Workshop on Silicon Architectures for Neural Net - 12 Vol de Vence - France, in Press.

[Brazil and Mandl 1990] Brazil, H. and E. Mandl. BPCS Optimized for Faster and Automated Speech Learning: Proceedings of the International Neural Network Conference (INNC 90 - Vol - Main - France, 987–721.

[Brazil and Grandoni 1991] Brazil, H. and M. Grandoni. Learning: Learning Computer Speech Learning with Memories Consideration for Neural Net Models. In Natural Computing: Trojans, Methods and Applications, P. Masera and A. Mail, eds. Elsevier, Amsterdam.

[Brazil and Cole 1988] Brazil, E. and H. Cole. A neural-net riemann mapping based on coupled first-photon mission. Oregon Graduate Center, CSE 88-410.

[Becker and le Cun 1988] Becker, S. and Y. le Cun. Improving the convergence of back-propagation learning with second-order methods. Proc. 1988 Connectionist Models Summer School, D. Touretzky, G. Hinton and T. Sejnowski, eds. . Morgan Kaufmann, San Mateo, CA.

[Barzo and Moore 1988] Barzo, Y. and B. Moore. Acceleration of learning. Proceedings CMCB-CMU School Trento - Italy.

[Bingham 1988] Bingham, J. A. C. The theory and Practice of Modern Design. Wiley, New York, NY.

[Broomhead and Powe 1988] Broomhead, D.S. and D. Powe. Multivariable functional interpolation and adaptive networks. Complex Systems 2, 155–225.

[Broyden et la. 1781] Broyden, C.G., J.E. Dennis, and L.L.Moir. On the local and superlinear convergence of quasi-Newton methods. IL.M.A. 12, 225–255.

[Carlon et la. 1991] Carlon, H., R. man Hookestatman and P. Dexpe. A Neural Network Application in Geophysics: Identification of Genetic Factor. Proceedings of the International Conference on Artificial Neural Networks (ICANN-19) - Espoo - Finland, 1521–9121.

[Care and Falmlee 1981] Care, W.L. and T. Falmlee. An adaptive riemann algorithm for back propagation networks. Computer Speech and Language 2, 815–205.

[Care 1991] Care, L. W. Efficacy of highly different learning algorithms of the back-propagation network. Proceedings IEEE TENCON-90.

[Dennis and Schnabel 1981] Dennis, J. E. and R. B. Schnabel. Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice Hall, Englewood Cliffs, NJ.

[Dennis 79 1981 ia. 1981] Dennis, J.E., J.J. Moré, and G.W. Stewart. Algorithm 573 NL2SOL—An adaptive nonlinear least-squares algorithm [E4]. TOMS 7, 369–383.

[Donahue 79 1981 ia. 1981] Donahue, J.L., C.B. Noe, R.L. Brandt and G.W. Stewart. LINPACK Users' Guide. 2nd, Philadelphia.

[Dyer and Hall 1991] Dyer, C. F. and S. Hall. Parallel multifrontal solvers for implicit acoustic finite element models. In Proceedings of the 5th International Conference on Computing Methods in Applied Mechanical Engineering (5). Boston - Dalian, 1331–1342.

[Kaufman 1993] Kaufman, L. An Empirical study of Levenberg-Marquardt Algorithms. Pitman, 231-23-JMO.

[Crawford and Brown 1991] Crawford, F. and D. Brown. Stochastic Algorithms and Multiple Threaded Programming: The Genetic Algorithm. Complex systems 4, 15–30.

[Hill 1981 ia. 1981] Hill, P.E., W. Murray and M.H. Wright. Practical Optimization. Academic Press, London.

[Glover 1991] Glover, F. Tabu Search Method in Artificial Intelligence and Operations Research. OR 2A 4, 14. 3, 5 or 9.

[Goldfarb 1976] Goldfarb, D. Factorized variable metric methods for unconstrained optimization. Math. Comp. 30, 118–135.

[Goldstein 1961] Goldstein, A.A. Constructive Real Analysis. Harper and Row, New York, NY.

[Jacobs 1988] Jacobs, M.A. Increased rates of convergence through learning rate adaptation. Neural Networks 1, 295–307.

[Leonard et al. 1990] Leonard, J.L., M.A. Kramer, L.H. Ungar, J.W. Davis, and R.D. Rippin. Comparison of Feed-Forward Neural Network Paradigms with Conjugate Gradient Method. Pressence University Montreal Canada, 1990-020-01-CL-JUV, Tertiary, Tertiary.

[Kollias and Anastassiou 1989] Kollias, S. and D. Anastassiou. An Adaptive Least Squares Algorithm for the Efficient Training of Multilayered Networks. IEEE Trans. on CAS 36, 1092–1101.

[Kramer and Sangiovanni-Vincentelli 1989] Kramer, A.H., and A. Sangiovanni-Vincentelli. Efficient Parallel Learning Algorithms for Neural Networks. In Advances in Neural Information Processing Systems, Jov. 1, 40–47.

[Madsen Kaldmann, and Madsen, 1991] Madsen Kaldmann, and Madsen, CA.

[Padgett and Taylor 1989] Padgett, A. and R. Taylor. A Non-Optimizing Approximation Processor for Neural Networks. In Advances in Neural Information Processing Systems, Jov. 1, 328–336.

[Powell 1977] Powell, M.J.D. Restart procedures for the conjugate gradient method. Mathematical Programming 12, 241–254.

[Powell 1981] Powell, M.J.D. Approximation Theory and Methods. Cambridge University Press, Cambridge, 1981.

[Rumelhart and Hinton 1986] Rumelhart, D. G. Hinton, and R. Williams. Learning Internal Representations by Error Propagation. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1. Foundations. MIT Press, Cambridge, MA.

[Saad 1981] Saad, Y. Krylov subspace methods for solving large unsymmetric linear systems. Mathematics of Computation 37, 105–126.

[Saad and Schultz 1985] Saad, Y. and M.H. Schultz. Parallel implementations of preconditioned conjugate gradient methods. YALEU/DCS/RR-425, October 1985.

[Saad and Schultz 1986] Saad, Y. and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 7, 856–869.

[Schnabel et al. 1983] Schnabel, R.B., J.E. Koontz, and B.E. Weiss. A modular system of algorithms for unconstrained minimization. ACM Trans. Math. Softw. 9, 132–160.

[Schultz et al. 1986] Schultz, M.H., S.C. Eisenstat, and Y. Saad. Yale Sparse Matrix Package: I. The Symmetric Codes. Int. J. Num. Meth. Eng. 18, 1145–1151.

[Shanno 1978] Shanno, D.F. Conjugate gradient methods with inexact searches. Math. of OR 3, 244–256.

[Shanno and Phua 1980] Shanno, D.F. and K.H. Phua. Remark on algorithm 500 - a variable method routine for function minimization. ACM TOMS 6, 618–622.

[Sorensen 1980] Sorensen, D.C. Newton's method with a model trust region modification. Technical Report 80-15, Argonne National Laboratory, Argonne, IL.

[Stewart 1973] Stewart, G.W. Introduction to Matrix Computations. Academic Press, New York, NY.

[Stewart 1976] Stewart, G.W. The economical storage of plane rotations. Numer. Math. 25, 137–138.

[Stewart 1979] Stewart, G.W. A bidiagonalization algorithm for the sparse least squares problem. Technical Report TR-1164, University of Maryland, College Park, MD.

[Stewart 1987] Stewart, G.W. Collinearity and least squares regression. Statistical Science 2, 68–100.

[Stewart 1988] Stewart, G.W. Stochastic perturbation theory. SIAM Review 30, 579–610.

[Stewart 1989] Stewart, G.W. On the perturbation of LU and Cholesky factors. Technical Report TR-2322, University of Maryland, College Park, MD.

[Stewart 1990] Stewart, G.W. Perturbation bounds for the QR factorization of a matrix. SIAM J. Num. Anal. 14, 509–518.

[Stewart 1991] Stewart, G.W. Perturbation theory for the singular value decomposition. Technical Report TR-2539, University of Maryland, College Park, MD.

[Stewart 1992] Stewart, G.W. An updating algorithm for subspace tracking. IEEE Trans. on Signal Processing 40, 1535–1541.

[Stewart 1993] Stewart, G.W. On the early history of the singular value decomposition. SIAM Review 35, 551–566.

[Stewart and Sun 1990] Stewart, G.W. and J. Sun. Matrix Perturbation Theory. Academic Press, San Diego, CA.

[Sun 1991] Sun, J. Perturbation bounds for the Cholesky and QR factorizations. BIT 31, 341–352.

[Sun 1992] Sun, J. On perturbation bounds for the QR factorization. Technical Report UMINF-92.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 1995] Sun, J. Perturbation analysis of the pole-shifting problem. SIAM J. Control and Optimization 33, 1975–1989.

[Sun 1996] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-96.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 1997] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-97.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 1998] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-98.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 1999] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-99.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2000] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-00.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2001] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-01.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2002] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-02.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2003] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-03.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2004] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-04.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2005] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-05.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2006] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-06.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2007] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-07.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2008] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-08.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2009] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-09.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2010] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-10.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2011] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-11.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2012] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-12.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2013] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-13.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2014] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-14.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2015] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-15.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2016] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-16.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2017] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-17.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2018] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-18.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2019] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-19.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2020] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-20.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2021] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-21.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2022] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-22.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2023] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-23.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2024] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-24.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2025] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-25.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2026] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-26.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2027] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-27.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2028] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-28.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2029] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-29.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2030] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-30.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2031] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-31.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2032] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-32.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2033] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-33.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2034] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-34.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2035] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-35.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2036] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-36.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2037] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-37.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2038] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-38.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2039] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-39.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2040] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-40.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2041] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-41.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2042] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-42.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2043] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-43.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2044] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-44.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2045] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-45.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2046] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-46.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2047] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-47.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2048] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-48.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2049] Sun, J. Perturbation analysis of the matrix inverse square root and polar decomposition. Technical Report UMINF-49.05, Institute of Information Processing, University of Umeå, Sweden.

[Sun 2050] Sun, J. Perturbation analysis of the matrix sign function. Technical Report UMINF-50.05, Institute of Information Processing, University of Umeå, Sweden.

[Malekzadeh et al., 2001] M. Malekzadeh, L. R. Perez, and G. Ayoubi. Using Neural Networks for Image Analysis in Oil Well Drilling. In Proceedings of the 11th International Workshop on Database Architectures and Neural Networks - Volume 1, pages 123–456, Muscat, Oman, 2001. Morgan Kaufmann.

[Miller, 2001] M. L. Miller. A Sorted Combinatorial Algorithm for Fast Improved Learning. Technical Report TR-222, Computer Science Department, University of Arizona, Tucson, 2001.

[Moss et al., 2001] M. Moss, L. L. Williams, R. S. Carlson, and K. E. Hillstrom. User Guide for MINARCH-1: Arizona National Radar Report DNA-01-4, 2001.

[Nocedal, 2001] Nocedal, J. Updating Quasi-Newton Matrices with Limited Storage. Mathematics of Computation, 35:773–782, 2001.

[Parker, 2001] Parker, D.B. Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order Hebbian learning. Proc. ICNN - 1 San Diego, CA, volume II – 593–II – 600, 2001.

[Porcello and Ghost, 2001] Porcello, T. and T. Ghost. Hexahistazine Algorithms for Learning Unit Size Equivalent to Multilayer Networks. Science, 294:512–516, 2001.

[Perez et al., 2001] Perez, W.W., B.B. Fernsby, A.A. Tenkolski, and T.W. Wright. Numerical Recipes in C: Cambridge University Press, Cambridge, MA.

[Hamilton et al., 2001] Hamilton, D.E., L. L. McClelland (eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations. TIM Press, Cambridge, MA.

[Schwenker and Lightman, 2001] Schwenker, L.T. and J.D. Lightman. TEMPEST: a battle network unit sensor of land mode. TR 1042, Hoban University, 33 and 62 Technical Report TUH\EE-88-10.

[Gnarr et al., 2001] Gnarr, T.W., L. A. Flanagan. Acceleration Techniques for the Backpropagation Algorithm. In Proceedings of the 8th International Conference on Artificial Neural Networks (ICANN-91 - Espoo - Finland), pages 828–828. North Holland, Amsterdam.

[Tomi, 2001] Tomi, L. On Large Scale Monitoring Case Study Clustering. MAS 1. Sci. Stat. Comput. 8(2), 314–319.

[Vincent, 2001] Vincent, I.M. Facial Feature Location in Coarse Resolution Images by Multi-Layer Perceptrons. In Proceedings of the 11th International Conference on Artificial Neural Networks (ICANN-91 - Espoo - Finland), pages 828–828. North Holland, Amsterdam.

[Yoy et al., 2001] Yoy, T.L., L.K. Manke, K.K. Riker, W. T. Smith and D. L. Allon. Acceleration the Convergence of the Back-Propagation Algorithm. Biological Cybernetics 62, 525–725.

[Watson, 2001] Watson, R. Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization. Technical Report TR-12-87-12, Univ. of Penn.

[Webb et al., 2001] Webb, A.R., D. Lowe and M.D. Bedworth. A Comparison of Non-linear Optimization Strategies for Adaptive Feed-forward Layered Networks. RSRE MEMO 4773, Hotel Sigma and Hadat Establishment, Malvern, England.

[Wolfe, 2001] Wolfe, H. Learning in Artificial Neural Networks: a Statistical Perspective. Neural Computation 1, 425–464.

[Watson and Seamus, 2001] Watson, B. and D.B. Seamus. Adaptive Signal Processing. Prentice Hall, Englewood Cliffs, NJ.