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

Roberto Battiti
Dipartimento di Matematica, Universitá di Trento
38050 Povo (Trento) - Italy
e-mail: battiti@itnvax.cineca.it

Revised: September 14, 1991

Abstract

On-line first order backpropagation is sufficiently fast and effective for many large-scale classification problems but for very high precision mappings, batch processing may be the method of choice. This paper reviews first- and second-order optimization methods for learning in feedforward neural networks. The viewpoint is that of optimization: many methods can be cast in the language of optimization techniques, allowing the transfer to neural nets of detailed results about computational complexity and safety procedures to ensure convergence and to avoid numerical problems. The review is not intended to deliver detailed prescriptions for the most appropriate methods in specific applications, but to illustrate the main characteristics of the different methods and their mutual relations.

1 Introduction

There are cases in which learning speed is a limiting factor in the practical application of multilayer perceptrons to problems that require high accuracy in the network mapping function. In this class are applications related to system identification and nonlinear modeling, time-series prediction, navigation, manipulation and robotics. In addition, the standard batch backpropagation (BP) method (e.g. [Rumelhart et al. 1986]) requires a selection of appropriate parameters by the user, that is mainly executed with a trial-and-error process. Since one of the competitive advantages of neural networks is the ease with which they may be applied to novel or poorly understood problems, it is essential to consider automated and robust learning methods with a good average performance on many classes of problems.

This review describes some methods that have been shown to accelerate the convergence of the learning phase on a variety of problems, and suggests a possible "taxonomy" of the different techniques based on their order (i.e., their use of first or second derivatives), space and computational requirements and convergence properties. Some of these methods, while requiring only limited modifications of the standard BP algorithm, yield a speed-up of very large factors[1] and, furthermore, are easier to apply because they do not require the choice of critical parameters (like the learning rate) by the neural network practitioners.

[1] It is not unusual to observe speed-ups of 100 - 1000 with respect to BP with fixed learning and momentum rates.

The presentation attempts a classification of methods derived from the literature based on their underlying theoretical frameworks, with particular emphasis on techniques that are appropriate for the supervised learning of multilayer perceptrons (MLPs).

The general strategy for the supervised learning of an input-output mapping is based on combining a quickly convergent local method with a globally convergent one. Local methods are based on appropriate *local models* of the function to be minimized. In the following sections, first we consider the properties of methods based on a *linear model* (steepest descent and variations), then we consider methods based on a *quadratic model* (Newton's method and approximations).

2 Backpropagation and Steepest Descent

The problem of learning an input-output mapping from a set of P examples can be transformed into the minimization of a suitably defined *error function*. Although different definitions of the error have been used, for concreteness we consider the "traditional" sum-of-squared-differences error function defined as:

$$E(w) = \frac{1}{2}\sum_{p=1}^{P} E_p = \frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{n_o}(t_{pi} - o_{pi}(w))^2 \quad (1)$$

where t_{pi} and o_{pi} are the target and the current output values for pattern p, respectively, and n_o is the number of output units. The learning procedure known as backpropagation [Rumelhart et al. 1986] is composed of two stages. In the first, the contributions to the gradient coming from the different patterns ($\partial E_p/\partial w_{ij}$) are calculated "backpropagating" the error signal. The partial contributions are then used to correct the weights, either after every pattern presentation (*on-line* BP), or after they are summed in order to obtain the total gradient (*batch* BP).

Let us define as g_k the gradient of the error function ($g_k = \nabla E(w_k)$). The batch backpropagation update is a form of gradient descent defined as:

$$w_{k+1} = w_k - \epsilon\, g_k \quad (2)$$

while the on-line update is:

$$w_{k+1} = w_k - \epsilon\, \nabla E_p(w_k) \quad (3)$$

If the *learning rate* ϵ tends to zero, the difference between the weight vectors w_{k+p} during one epoch of the on-line method tends to be small and the step $\epsilon\nabla E_p(w_{k+p})$ induced by a particular pattern p can be approximated by $\epsilon\nabla E_p(w_k)$ (by calculating the gradient at the initial weight vector). Summing the contributions for all patterns, the movement in weight space during one epoch will be similar to the one obtained with a single batch update. However in general the *learning rate* has to be large to accelerate convergence, so that the paths in weight space of the two methods differ.

The on-line procedure *has* to be used if the patterns are not available before learning starts (see for example the perceptron used for adaptive equalization in [Widrow and Stearns 1985]), and a continuous adaptation to a stream of input-output signals is desired. On the contrary, if all patterns are available, collecting the total gradient information before deciding the next step can be useful in order to avoid a mutual interference of the weight changes (caused by the different patterns) that could occur for large *learning rates* (this effect is equivalent to a sort of noise in the true gradient direction). One of the reasons in favor of the on-line approach is that it possesses some *randomness* that may help in escaping from a local minimum. The objection to this is that the method may, for the same reason, *miss* a good local minimum, while there is the alternative method of converging to the "nearest" local minimum and using randomness[2] to escape only after convergence. In addition, the on-line update may be useful when the number of patterns is so large that the errors involved in the *finite precision* computation of the total gradient may be comparable with the gradient itself. This effect is particularly present for analog implementations of backpropagation, but it can be controlled in digital implementations by increasing the number of bits during the gradient accumulation. The fact that many patterns possess redundant information (see for example the case of hand-written character recognition in [le Cun 1986]) has been cited as an argument in favor of on-line BP, because many of the contributions to the gradient are similar, so that waiting for all contributions before updating can be wasteful. In other words, averaging over more examples to obtain a better estimate of the gradient does not improve the learning speed sufficiently to compensate for the additional computational cost of taking these patterns into account. Nonetheless, the redundancy can also be limited using batch BP, provided that learning is started with a subset of "relevant" patterns and continued after convergence by progressively increasing the example set. This method has for example been used in [Kramer and Sangiovanni-Vicentelli 1988] for the digit recognition problem[3]. Even if the training set is redundant, on-line BP may be slow relative to second-order methods for badly conditioned problems. The convergence of methods based on gradient descent (or approximations thereof) depends critically on the relative size of the maximum and minimum eigenvalues of the Hessian matrix (see [le Cun et al. 1991] and eqn. 6 for the case of steepest descent). This is related to the "narrow valley" effect described in [Rumelhart et al. 1986]. In addition, the batch approach lends itself to straightforward modifications using second-order information, as it will be shown in the following sections.

At this point, in order not to mislead the reader in the choice of the most appropriate method for a specific application, it is useful to remember that many large-scale experiments (mainly in pattern recognition - classification) have used the simple on-line version of BP with full satisfaction, considering both the final result and the number of iterations. In some cases, with a careful tuning of the on-line procedure the solution is reached in a very small number of *epochs*, i.e. in a few presentations of the complete example set, (see for example [Rumelhart et al. 1986]) and it is difficult to reach a comparable learning speed with batch techniques [Cardon et al. 1991]. Tuning operations are, for example, the choice of appropriate parameters like the learning and momentum rate [Fahlman 1988], 'annealing' schedules for the learning rate (that is progressively reduced) [Malferrari et al. 1990], updating schemes based on summing the contributions of related patterns [Sejnowski and Rosenberg 1986], "small batches", "selective" corrections only if the error is larger than a threshold (that may be progressively reduced) [Vincent 1991], [Allred and Kelly 1990], randomization of the sequence of pattern presentation, etc. The given references are only some examples of significant applications, out of an abundant literature[4]. Because in these cases only approximated output values are required and the example set often is characterized by a large degree of redundancy, these two attributes could be considered as votes in favor of on-line BP, again

---

[2] In addition, there are good reasons why random noise may not be the best available alternative to escape from a local minimum and avoid returning to it. See, for example, the recently introduced *TABU* methods [Glover 1987].

[3] If the redundancy is clear (when for example many copies of the *same* example are present) one may pre-process the example set in order to eliminate the duplication. On the contrary, if redundancy is only partial, the redundant patterns have to be presented to and learned by the network in *both* versions.

[4] The availability of many variations of the on-line technique is one of the reasons why 'fair' comparisons with the batch and second-order versions are complex. Which version has to be chosen for the comparison? If the final convergence results have been obtained after a tuning process, should the tuning times be included in the comparison?

provided that the trial-and-error 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 arbitrary way, there is no guarantee that the net will converge to a point with vanishing gradient. Nonetheless, convergence in the on-line approach can be obtained by appropriately choosing a fixed and sufficiently small *learning rate*.

The issue of an appropriate fixed learning rate for on-line LMS learning has been investigated in the adaptive signal processing literature (see for example [Bingham 1988]). The result is that the convergence of stochastic LMS is guaranteed if $\epsilon < 1/(N\ \sqrt{\lambda_{max}})$, where N is the number of parameters being optimized and λ_{max} is the largest eigenvalue of the auto-correlation function of the input[5]. A detailed study of adaptive filters is presented in [Widrow and Stearns 1985]. The effects of the autocorrelation matrix of the inputs on the learning process (for a sigle linear unit) are discussed in [le Cun et al. 1991]. In this framework the appropriate learning rate for gradient descent is $1/\lambda_{max}$. These results cannot be extended to multilayer networks (with non-linear transfer functions) in a straightforward manner, but they can be used as starting point for useful heuristics.

The convergence properties of the LMS algorithm with *adaptive* learning rate are presented in [Luo 1991], together with a clear comparison of the LMS algorithm with stochastic gradient descent and adaptive filtering algorithms. The main result is that if the learning rate ϵ_n for the nth training cycle satisfies the two conditions:

$$\sum_{n=1}^{\infty} \epsilon_n = \infty \qquad \sum_{n=1}^{\infty} \epsilon_n^2 < \infty \tag{4}$$

then the sequence of weight matrices generated by the LMS algorithm (with a cyclic pattern presentation) will converge to the optimal solution (minimizing the mean-square error).

But even if ϵ is appropriately chosen so that the error decreases with a reasonable speed and oscillations are avoided, gradient descent *is not* always the fastest method to employ. This is not an intuitive result, because the negative gradient is the direction of fastest decrease in the error. Unfortunately, the "greed" in trying to reach the minimum along this one-dimensional direction is paid at the price that subsequent gradient directions tend to interfere, so that in a weight space with dimension N, the one-dimensional minimization process has to be repeated for a number of times that is normally much higher than N (even for simple quadratic functions)[6]. In the steepest descent method, the process of minimizing along successive negative gradients is described as:

$$\begin{aligned} w_{k+1} &= w_k - \epsilon_k g_k \\ \text{where} \quad \epsilon_k &\quad \text{minimizes} \quad E(w_k - \epsilon g_k) \end{aligned} \tag{5}$$

If steepest descent is used to minimize a quadratic function $Q(w) = c^T w + \frac{1}{2} w^T G w$ (G symmetric and positive definite), it can be shown that:

$$|Q(w_{k+1}) - Q(w_*)| \approx \left(\frac{\eta_{max} - \eta_{min}}{\eta_{max} + \eta_{min}}\right)^2 |Q(w_k) - Q(w_*)| \tag{6}$$

[5] I owe this remark to the referee.

[6] It is easy to show that, if exact one-dimensional optimization along the negative gradient is used, the gradient at the next step is *perpendicular* to the previous one. If, considering an example in two dimensions, the lines at equal error are given by elongated ellipsoids, the system, for a "general" starting point, goes from a point to the one that is tangent to the equal-error lines along the gradient, and then repeats along a perpendicular direction.

where η_{max} and η_{min} are the maximum and minimum eigenvalues of the matrix G, and w_* is the minimizer (see [Luenberger 1973]). If these two eigenvalues are very different, the distance from the minimum value is multiplied each time by a number that is close to one. The type of convergence in eqn. 6 is termed *q-linear convergence*. One has *q-superlinear convergence* if, for some sequence c_k that converges to 0, one has

$$|Q(w_{k+1}) - Q(w_*)| \leq c_k |Q(w_k) - Q(w_*)| \tag{7}$$

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

$$|Q(w_{k+1}) - Q(w_*)| \leq c|Q(w_k) - Q(w_*)|^p \tag{8}$$

In practice *q-linear* convergence tends to be very slow, while *q-superlinear* or *q-quadratic* ($p = 2$) convergence is eventually much faster[7].

In the following sections, we illustrate some techniques that can be used to ensure convergence, to avoid numerical problems related to finite-precision computation, and to accelerate the minimization process with respect to standard batch BP.

3 Conjugate Gradient Methods

Let us consider a quadratic function $Q(w)$ of the type described in the previous section. We have seen that one of the difficulties in using the steepest descent method is that a one-dimensional minimization in direction a followed by a minimization in direction b did *not* imply that the function was minimized on the subspace generated by a and b. Minimization along direction b may in general *spoil* a previous minimization along direction a (this is why the one-dimensional minimization in general has to be repeated a number of times much larger than the number of variables). On the contrary, if the directions were *non-interfering* and linearly independent, at the end of N steps the process would converge to the minimum of the quadratic function.

The concept of *non-interfering* directions is at the basis of the conjugate gradient method (CG) for minimization. Two directions are *mutually conjugate* with respect to the matrix G if:

$$p_i^T G p_j = 0 \quad \text{when} \quad i \neq j \tag{9}$$

After minimizing in direction p_i, the gradient at the minimizer will be perpendicular to p_i. If a second minimization is in direction p_{i+1}, the change of the gradient along this direction is $g_{i+1} - g_i = \alpha G p_{i+1}$ (for some constant α). The matrix G is indeed the Hessian, the matrix containing the second derivatives, and in the quadratic case the model coincides with the original function. Now, if eqn. 9 is valid, this change is *perpendicular* to the previous direction ($p_i^T(g_{i+1} - g_i) = 0$), therefore the gradient at the new point *remains* perpendicular to p_i and the previous minimization is not spoiled. While for a quadratic function the conjugate gradient method is guaranteed to converge to the minimizer in at most $(N + 1)$ function and gradient evaluations (apart from problems caused by the finite precision), for a general function it is necessary to iterate the method until a suitable approximation to the minimizer is obtained.

[7] *For example, for a q-quadratically convergent method, the number of significant digits in the solution is doubled after each iteration.*

Let us introduce the vector $y_k = g_{k+1} - g_k$. The first search direction p_1 is given by the negative gradient $-g_1$. Then the sequence w_k of approximations to the minimizer is defined in the following way:

$$w_{k+1} = w_k + \alpha_k p_k \tag{10}$$

$$p_{k+1} = -g_{k+1} + \beta_k p_k \tag{11}$$

(12)

where g_k is the gradient, α_k is chosen to minimize E along the search direction p_k and β_k is defined by:

$$\beta_k = \frac{y_k^T g_{k+1}}{y_k^T p_k} \tag{13}$$

There are different versions of the above equation, in particular, the Polak-Ribiere choice is $\beta_k = \frac{y_k^T g_{k+1}}{g_k^T g_k}$, the Fletcher-Reeves choice is $\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$; they all coincide for a quadratic function [Shanno 1978]. A major difficulty with all the above forms is that, for a general function, the obtained directions are *not* necessarily descent directions and numerical instability can result.

Although for a wide class of functions the traditional CG method with exact searches and exact arithmetic is *superlinearly convergent*, implementations of the conjugate-gradient method with finite precision computation are "nearly always linearly convergent" [Gill et al. 1981], but the number of steps is in practice much smaller than that required by steepest descent.

The use of a *momentum* term to avoid oscillations in [Rumelhart et al. 1986] can be considered as an approximated form of conjugate-gradient. In both cases, the gradient direction is modified with a term that takes the previous direction into account, the important difference being that the parameter β in conjugate-gradient is automatically defined by the algorithm, while the *momentum* rate has to be "guessed" by the user. Another difficulty related to the use of a *momentum* term is due to the fact that there is an upper bound on the adjustment caused by the momentum. For example, if all partial derivatives are equal to 1, then the exponentially weighted sum caused by the momentum rate α converges to $1/(1-\alpha)$, see [Jacobs 1988] for details[8]. Furthermore, summing the momentum term to the one proportional to the negative gradient may produce an *ascent* direction, so that the error *increases* after the weight update.

Among the researchers using conjugate gradient methods for the MLP are [Barnard and Cole 1988], [Johansson et al. 1990], [Bengio and Moore 1989], [Drago and Ridella 1991], Hinton's group in Toronto, the groups at CMU, Bell Labs, etc. A version in which the one-dimensional minimization is substituted by a scaling of the step that depends on success in error reduction and goodness of a one-dimensional quadratic approximation is presented in [Moller 1990] (SCG). This $O(N)$ scheme encorporates ideas from the "model-trust region" methods (see section 4.3) and "safety" procedures that are absent in the CG schemes, yielding convergence results that are comparable with the OSS method described in section 6. It is worth stressing that expensive one-dimensional searches are also discouraged by current results in optimization (see section 4.1): the search *can* be executed using only a couple of function and gradient evaluations.

[8] In addition, B. Pearlmutter has recently shown that momentum even if chosen 'optimally' can do no better than q-linear convergence, see his poster at the coming NIPS 1991 conference.

4 Newton's Method

Newton's method can be considered as the basic local method using second-order information. It is important to stress that its practical applicability to multilayer perceptrons is hampered by the fact that it requires a calculation of the Hessian matrix (a complex and expensive task[9]). Nonetheless, the method is briefly illustrated because most of the "useful" second-order methods originate from it as approximations or variations.

It is based on modeling the function with the first three terms of the Taylor-series expansion about the current point w_c:

$$E(w_c+s) \approx m_c(w_c+s) \stackrel{\text{def}}{=} E(w_c) + \nabla E(w_c)^T s + \frac{1}{2} s^T \nabla^2 E(w_c) s \tag{14}$$

and solving for the step s^N that brings to a point where the gradient of the model is zero: $\nabla m_c(w_c + s^N) = 0$. This corresponds to solving the following linear system:

$$\nabla^2 E(w_c) s^N = -\nabla E(w_c) \tag{15}$$

s^N is , by definition, Newton's step (and direction).

If the Hessian matrix ($\nabla^2 E$ or H, for short) is positive definite and the quadratic model is correct, one iteration is sufficient to reach the minimum. Because one iteration consists in solving the linear system in eqn. 15, the complexity of one step is $O(N^3)$, using standard methods[10]. In general, if the initial point w_0 is sufficiently close to the minimizer w_*, and $\nabla^2 E(w_*)$ is positive definite, the sequence generated by repeating Newton's algorithm converges *q-quadratically* to w_* (see [Dennis and Schnabel 1983] for details).

Assuming that the Hessian matrix can be obtained in reasonable computing times, the main practical difficulties in applying the "pure" Newton's method of eqn. 15 arise when the Hessian is not positive definite, or when it is singular or ill-conditioned. If the Hessian is not positive definite (this may be the case in multilayer perceptron learning!), there is no "natural" scaling in the problem: there are directions p_k of negative curvature (i.e., such that $p_k^T H p_k \leq 0$) that would suggest "infinite" steps in order to minimize the model. Unfortunately, long steps increase the probability of leaving the region where the model is appropriate, producing nonsense. This behavior is not uncommon for multilayer perceptron learning: in some cases a local minimization step increases some weights by large amounts, pushing the output of the sigmoidal transfer function into the saturated region. When this happens, some second derivatives are very small and, given the finite machine precision or the approximations, the calculated Hessian will not be positive definite. Even if it is, the linear system of eqn. 15 may be seriously ill-conditioned. *Modified Newton's methods* encorporate techniques for dealing with the above problems, changing the model Hessian in order to obtain a sufficiently positive definite and non-singular matrix.

It is worth observing that, although troublesome for the above reasons, the existence of directions of negative curvature may be used to continue from a saddle point where the gradient is close to zero[11].

[9] A "brute force" method to calculate H is that of using a finite-difference formula. If the gradient is available (as is the case for feed-forward nets), one may use: $H_{\cdot j} = (\nabla E(w_c + h_j e_j) - \nabla E(w_c))/h_j$, with suitable h_j steps, see [Dennis et al. 1981]. Note that $N+1$ gradient computations are needed, so that the method is not suggested for large networks!

[10] A smaller upper bound for matrix inversion is actually $O(n^{\log_2 7})$, see [Press et al. 1988] for details.

[11] Only second-order methods provide this possibility, while methods based on steepest descent are condemned to failure in this case: how many "local minima" are in reality saddle points!

While calculating the analytic gradient for multilayer perceptrons can be efficiently executed "backpropagating" the error, calculating the Hessian is computationally complex, so that practical methods have to rely onto suitable approximations. In the following sections we illustrate some modifications of Newton's method to deal with global convergence, indefinite Hessian and iterative approximations to the Hessian itself. In the review by [White 1989] the use of appropriate modifications of Newton's methods for learning is considered starting from a statistical perspective.

4.1 Globally Convergent Modifications: Line Searches

Considering second-order methods, their fast *local* convergence property suggests that they are used when the approximation is *close* to the minimizer. On the other hand, getting from an initial configuration (that may be very different from the minimizer) to a point where the local model is accurate requires some additional effort. The key idea to obtain a general-purpose successful learning algorithm is that of combining a fast *tactical* local method with a robust *strategic* method that assures global convergence.

Because Newton's method (or its modifications when the analytic Hessian is not available) has to be used near the solution, the suggested method is that of trying the Newton step *first*, deciding whether the obtained point is acceptable or not, and *backtracking* in the last case (i.e. selecting a shorter step in the Newton direction).

One of the reasons for searching 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:

$$\frac{dE}{d\lambda}(w_c + \lambda s^N) = \nabla E(w_c)^T s^N = -\nabla E(w_c)^T H_c^{-1} \nabla E(w_c) < 0$$

If the analytic Hessian has to be approximated, it is essential to consider only approximations that maintain *symmetry* and *positive definiteness*, so that the one-dimensional minimization step remains unaltered.

When using a line-search algorithm outside the convergence region of Newton's method, some simple prescriptions have to be satisfied in order to obtain global convergence. Let the accepted step along the search direction p_c be λ_c, the requirement that $E(w_c+\lambda_c p_c) < E(w_c)$ is not sufficient: the sequence w_k 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 (I) and they must be long enough to ensure sufficient progress (II). Furthermore, the search direction must be kept sufficiently away form being orthogonal to the gradient.

A formulation of the above requirements that is frequently used in optimization is based on work of Armijo and Goldstein (see for example [Goldstein 1967]). Requirement (I) and (II) become:

$$E(w_c + \lambda_c p) \;\leq\; E(w_c) + \alpha\lambda_c \nabla E(w_c)^T p \tag{16}$$

where α is a fixed constant $\in (0, 1)$ and $\lambda > 0$,

$$\nabla E(w_c + \lambda_c p)^T p \;\geq\; \beta \nabla E(w_c)^T p \tag{17}$$

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

If the two above conditions are satisfied at each iteration and if the error is bounded below, the sequence w_k obtained is such that $\lim_{k \to \infty} \nabla E(w_k) = 0$, provided that each step is kept away from orthogonality to the gradient ($\lim_{k \to \infty} \frac{\nabla E(w_k)^T s_k}{\|s_k\|} \neq 0$).

This result is quite important: we are permitted to use *fast* approximated one-dimensional searches without losing global convergence. Recent computational tests show that methods based on fast one-dimensional searches in general require much less computational effort than methods based on sophisticated one-dimensional minimizations[12].

The line-search method suggested in [Dennis and Schnabel 1983] is well suited for multilayer perceptrons (where the gradient can be obtained with limited effort during the computation of the error) and requires only a couple of error and gradient evaluations per iteration, in the average. The method is based on quadratic and cubic interpolations and designed to use in an efficient way the available information about the function to be minimized (see [Dennis and Schnabel 1983] for details). A similar method based on quadratic interpolation is presented in [Battiti 1989].

4.2 Indefinite Hessians: Modified Cholesky Factorization

When the Hessian matrix in the local model introduced in eqn. 14 is not positive definite and well conditioned, eqn. 15 cannot be used without modifications. This can be explained by introducing the spectral decomposition of the Hessian (based on the availability of eigenvalues and eigenvectors), writing the matrix H as a sum of projection operators:

$$H = U \Lambda U^T = \sum_{i=1}^{N} \eta_i u_i u_i^T \tag{18}$$

where Λ is diagonal (Λ_{ii} is the eigenvalue η_i) and U orthonormal. It is easy to see that, if some eigenvalues are close to zero (with respect to the largest eigenvalue), the inverse matrix has eigenvalues close to infinity, a sure source of numerical problems[13]. If one eigenvalue is negative, the quadratic model does *not* have a minimum, because large movements in the direction of the corresponding eigenvector decrease the error value to arbitrarily negative values.

A recommended strategy for changing the model Hessian in order to avoid the above problems is that of summing to it a simple diagonal matrix of the form $\mu_c I$ (I being the identity matrix), so that $(\nabla^2 E(w_c) + \mu_c I)$ is positive definite and safely well conditioned. A proper value for μ_c can be efficiently found using the modified Cholesky factorization described in [Gill et al. 1981] and the heuristics described in [Dennis and Schnabel 1983]. The resulting algorithm is as follows.

The Cholesky factors of a positive-definite symmetric matrix can be considered as a sort of "square root" of the matrix. The original matrix M is expressed as the product LDL^T, where L is a *unit lower triangular* matrix[14] and D is a diagonal matrix with strictly positive diagonal elements. Taking the square root of the diagonal elements and forming with them the matrix $D^{\frac{1}{2}}$,

---

[12] In simple words: it does not pay to use a method that requires a limited number of iterations if each iteration requires a huge amount of computation.

[13] In detail, the conditioning number $\kappa(A)$ of a matrix A is defined as $\|A\|\|A^{-1}\|$, where $\|\ \|$ is the matrix operator norm induced by the vector norm. The conditioning number is the ratio of the maximum to the minimum stretch induced by A and measures, among other effects, the sensitivity of the solution of a linear system to finite-precision arithmetic, that is of the order of $\kappa(A)(machine precision)$. $\kappa(A)$ does not depend on scaling of the matrix by a fixed constant.

[14] A *unit lower triangular* matrix has the diagonal elements equal to one and all the elements above the diagonal equal to zero.

the original matrix can be written as $M = LD^{\frac{1}{2}}D^{\frac{1}{2}}L^T = \overline{L}\,\overline{L}^T = R^T R$, where $\overline{L}$ is a general lower triangular matrix, and R a general 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 stability*. If the original matrix is *not* positive definite, the factorization can be modified in order to obtain factors L and D with all the diagonal elements in D positive and all the elements in L uniformly bounded. The obtained factorization corresponds to the factors of a matrix $\overline{H}_c$ differing from the original one only by a diagonal matrix K with non-negative elements:

$$\overline{H}_c = LDL^T = H_c + K \tag{19}$$

$$\text{where} \quad d_k > \delta \; ; \; |l_{ij}|\sqrt{d_k} \leq \beta \tag{20}$$

A suitable choice of β is described in [Gill et al. 1981] [15].

The availability of the modified Cholesky factorization is the starting point for the algorithm described in [Dennis and Schnabel 1983] to find $\mu_c \geq 0$ such that $\nabla^2 E(w_c) + \mu_c I$ is positive definite and well conditioned. Considering the eigenspace of $\nabla^2 E(w_c)$, μ_c has to be slightly larger than the magnitude of the most negative eigenvalue. If the matrix E is zero, μ_c is set to zero, otherwise it is set to the minimum of two upper bounds. One upper bound is:

$$u_1 = \max_{1 \leq i \leq N} \{k_{ii}\}$$

In fact, the magnitude of the most negative eigenvalue of $\nabla E(w_c)$ must be less that the maximum k_{ii}, because after summing k_{ii} to it the eigenvalue becomes positive (remember that the modified factorization produces a positive definite matrix). The other upper bound is derived from the Gerschgorin circle theorem and is defined as:

$$u_2 = -\min_{1 \leq i \leq N} \{h_{ii} - \sum_{j=1; j \neq i}^{N} |h_{ij}|\} + \sigma$$

where σ is a positive factor needed to take the finite precision of computation into account[16].

If u_2 is added to the diagonal elements, the matrix becomes *strictly diagonally dominant* ($h_{ii} - \sum_{j=1; j \neq i}^{n} |h_{ij}| > 0$) and therefore positive definite. The algorithm uses $\mu_c = \min\{u_1, u_2\}$.

Software routines for the Cholesky decomposition can be found in [Dennis and Schnabel 1983] or in [Dongarra et al. 1979].

4.3 Relations with Model-Trust Region Methods

Up to now we have considered optimization techniques based on finding a search *direction* and moving by an acceptable amount in that direction ("step-length-based methods"). In Newton's method the direction was obtained by 'multiplying' the gradient by the inverse Hessian, and the step was the full Newton step when possible (to obtain fast local convergence) or a shorter step when the global strategy required it. In section 4.2 we described methods to modify the Hessian

[15] The prescription is: $\beta^2 = \max\{\gamma, \xi/\sqrt{n^2 - 1}, \epsilon_M\}$, where γ is the largest in magnitude of the diagonal elements of H_c, ξ the largest in magnitude of the off-diagonal elements, and ϵ_M the machine precision. This result is obtained by requiring that the diagonal modification is *minimal* (minimal norm of K), and that sufficiently positive-definite matrices H_c are left unaltered (K null in this case).

[16] σ is $\sqrt{\epsilon_M}$ ($maxev - minev$), ϵ_M being the smallest positive number ϵ such that $1 + \epsilon > 1$ and $maxev$, $minev$ being estimates of the maximum and minimum eigenvalues.

when this was not positive definite. Because the modification consisted in adding to the local model a term quadratic in the step magnitude ($m_{modified}(w_c + s) = m_c(w_c + s) + \mu_c s^T s$), one may suspect that minimizing the new model is equivalent to minimizing the original one with the constraint that the step s is not too large. Now, while in line-search algorithms the direction is maintained and only the step length is changed, there are alternative strategies based on choosing first a step *length* and then using the *full* quadratic model (not just the one-dimensional one of eqn. 14) to determine the appropriate direction. These competitive methods are called "model-trust-region methods" with the idea that the model is trusted only within a region, that is updated using the experience accumulated during the search process.

The above suspicion is true and there is indeed a close relationship between "trust-region" methods and "line-search" methods with diagonal modification of the Hessian. This relationship is described by the following theorem. Suppose that we are looking for the step s_c that solves:

$$\min m_c(w_c + s) = E(w_c) + \nabla E(w_c)^T s + \frac{1}{2} s^T H_c s; \qquad \text{subject to} \quad ||s|| \leq \delta_c \tag{21}$$

the above problem is solved by:

$$s(\mu) = -(H_c + \mu I)^{-1}\, \nabla E(w_c) \tag{22}$$

for the unique $\mu \geq 0$ such that the step has the maximum allowed length ($||s(\mu)|| = \delta_c$), unless the step with $\mu = 0$ is inside the trusted region ($||s(0)|| \leq \delta_c$), in which case $s(0)$ is the solution, equal to the Newton step. We omit the proof and the usable techniques for finding μ, leaving the topics as a suggested reading for those in search of elegance and inspiration (see for example [Dennis and Schnabel 1983]).

As a final observation, note that the diagonal modification to the Hessian is a sort of compromise between gradient descent and Newton's method: When μ tends to zero the original Hessian is (almost) positive definite and the step tends to coincide with Newton's step; when μ has to be large the diagonal addition μI tends to dominate and the step tends to one proportional to the negative gradient:

$$s(\mu) = -(H_c + \mu I)^{-1} \nabla E(w_c) \approx -\frac{1}{\mu} \nabla E(w_c)$$

There is no need to decide from the beginning about using or not the gradient as a search direction; the algorithm takes care of selecting the direction that is appropriate for a local configuration of the error surface.

While not every usable multilayer perceptron needs to have thousands of weights, it is true that this number tends to be large for some interesting applications. Furthermore, while the analytic gradient is easily obtained, the calculation of second derivatives is complex and time-consuming.

For these reasons, the methods described above, while fundamental from a theoretical standpoint, have to be simplified and approximated in suitable ways that we describe in the next two sections.

5 Secant Methods

When the Hessian is not available analytically, secant methods[17] are widely used techniques for approximating the Hessian in an iterative way using only information about the gradient. In one dimension the second derivative $\frac{\partial^2 E(w)}{\partial w^2}$ can be approximated with the slope of the secant line (therefore the term "secant") through the values of the first derivatives in two near points:

$$\frac{\partial^2 E(w)}{\partial w^2}\,(w_2 - w_1) \approx \left(\frac{\partial E(w_2)}{\partial w} - \frac{\partial E(w_1)}{\partial w}\right) \tag{23}$$

In more dimensions the situation is more complex. Let the current and next point be w_c and w_+; defining $y_c = \nabla E(w_+) - \nabla E(w_c)$ and $s_c = w_+ - w_c$; the analogous secant equation to eqn. 23 is:

$$H_+ \, s_c = y_c \tag{24}$$

The new problem is that in more than one dimension eqn. 24 does not determine a unique H_+ but leaves the freedom to choose from a $(N^2 - N)$-dimensional affine subspace $Q(s_c, y_c)$ of matrices obeying eqn. 24. The new suggested strategy is that of using eqn. 24 not to *determine* but to *update* a previously available approximation. In particular, Broyden's update is based on a *least change* principle: find the matrix in $Q(s_c, y_c)$ that is closest to the previously available matrix. This is obtained by *projecting*[18] the matrix onto $Q(s_c, y_c)$. The resulting Broyden's update is:

$$(H_+)_1 = H_c + \frac{(y_c - H_c s_c) s_c^T}{s_c^T s_c} \tag{25}$$

Unfortunately, Broyden's update does not guarantee a *symmetric* matrix. For this reason, its use in optimization is strongly discouraged (unless we are willing to live with directions that are not descent directions, a basic requirements of line-search methods).

Projecting the matrix obtained with Broyden's update onto the subspace of *symmetric* matrices is not enough: the new matrix may be out of $Q(s_c, y_c)$. Fortunately, if the two above projections are repeated, the obtained sequence of matrices $(H_+)_n$ converges to a matrix that is *both* symmetric and in $Q(s_c, y_c)$. This is the *symmetric* secant update of Powell:

$$H_+ = H_c + \frac{(y_c - H_c s_c) s_c^T + s_c (y_c - H_c s_c)^T}{s_c^T s_c} - \frac{< y_c - H_c s_c, s_c > s_c s_c^T}{(s_c^T s_c)^2} \tag{26}$$

The Powell update is one step forward, but not the solution. In the previous sections we have shown the importance of having a *symmetric* and *positive definite* approximation to the Hessian. Now, one can prove that H_+ is symmetric and positive definite if and only if $H_+ = J_+ J_+^T$ for some nonsingular matrix J_+. Using this fact, one update of this kind can be derived, expressing $H_+ = J_+ J_+^T$ and using Broyden's method to derive a suitable J_+ [19]. The resulting update is historically known as the BFGS update (by Broyden, Fletcher, Goldfarb and Shanno in 1970) and is given by:

$$H_+ = H_c + \frac{y_c y_c^T}{y_c^T s_c} - \frac{H_c s_c s_c^T H_c}{s_c^T H_c s_c} \tag{27}$$

---

[17] Historically these methods were called *quasi-Newton* methods. Here we follow the terminology of [Dennis and Schnabel 1983], where the term *quasi-Newton* refers to all algorithms "derived" from Newton's method.

[18] The changes and the projections are executed using the Frobenius norm: $\|H\|_F = (\sum_{i,j} h_{ij}^2)^{\frac{1}{2}}$, the matrix is considered as a "long" vector.

[19] The solution exists if $y_c s_c > 0$, that is guaranteed if "accurate" line-searches are performed, see section 4.1.

The BFGS positive definite secant update has been the most successful update in a number of studies performed during the years. The positive definite secant update converges q-superlinearly (a proof can be found in [Broyden et al. 1973]). It is common to take the initial matrix H_0 as the identity matrix (first step in the direction of the negative gradient). It is possible to update directly the Cholesky factors [Goldfarb 1976], with a total complexity of $O(N^2)$ (see the implementation in [Dennis and Schnabel 1983]). Secant methods for learning in the multilayer perceptron have been used for example in [Watrous 1987]. The $O(N^2)$ complexity of BFGS is clearly a problem for very large networks, but the method can still remain competitive if the number of examples is very large, so that the computation of the error function dominates. A comparison of various non-linear optimization strategies can be found in [Webb et al. 1988]. Second-order methods in continuous time are considered in [Parker 1987].

6 Closing the Gap: Second-Order Methods with O(N) Complexity

One drawback of the BFGS update of eqn. 27 is that it requires storage for a matrix of size $N \times N$ and a number of calculations of order $O(N^2)$ [20]. Although the available storage is less of a problem now than it was a decade ago (for a possible method to cope with limited storage, see for example [Nocedal 1980]), the computational problem still exists when N becomes of the order of one hundred or more.

Fortunately, it is possible to kill two birds with one stone. In [Battiti 1989] it is shown that it is possible to use a secant approximation with $O(N)$ computing and storage time that uses second-order information. This OSS method (*one-step secant*) does not require the choice of critical parameters, is guaranteed to converge to a point with zero gradient and has been shown to accelerate the learning phase by many orders of magnitude with respect to batch BP if high precision in the output values is desired [Battiti and Masulli 1990]. In cases where approximated output values are sufficient, the OSS method is usually better or comparable with "fair" versions of backpropagation[21].

While the term OSS should be preferred, historically OSS is a variation of what is called one-step (*memory-less*) *Broyden-Fletcher-Goldfarb-Shanno method*. In addition to reducing both the space and computational complexity of the BFGS method to $O(N)$, this method provides a strong link between secant methods and the *conjugate gradient* methods described in section 3.

The derivation starts by inverting eqn. 27, obtaining the *positive definite secant update* for the inverse Hessian:

$$H_+^{-1} = H_c^{-1} + \frac{(s_c - H_c^{-1}y_c)s_c^T + s_c(s_c - H_c^{-1}y_c)^T}{y_c^T s_c} - \frac{< s_c - H_c^{-1}y_c, y_c > s_c s_c^T}{(y_c^T s_c)^2} \qquad (28)$$

Now, there is an easy way to reduce the storage for the matrix H_c : just forget the matrix and start each time from the identity I. Approximating eqn. 28 in this way, and multiplying by the

[20] Updating the Cholesky factorization and calculating the solution are both of order $O(N^2)$. The same order is obtained if the *inverse* Hessian is updated, as in eqn. 28, and the search direction is calculated by a matrix-vector product.

[21] The comparison with BP with fixed learning and momentum rate has little meaning: if an improper learning rate is chosen, standard BP becomes arbitrarily slow or not convergent, if parameters are chosen with a slow trial-and-error process this time should also be included in the total computing time.

gradient $g_c = \nabla E(w_c)$, the new search direction p_+ becomes:

$$p_+ = -g_c + A_c s_c + B_c y_c \tag{29}$$

where the two scalars A_c and B_c are the following combination of scalar products of the previously defined vectors s_c, g_c and y_c (last step, gradient and difference of gradients):

$$A_c = -\left(1 + \frac{y_c^T y_c}{s_c^T y_c}\right)\frac{s_c^T g_c}{s_c^T y_c} + \frac{y_c^T g_c}{s_c^T y_c} \quad ; \quad B_c = \frac{s_c^T g_c}{s_c^T y_c}$$

The search direction at the beginning of learning is taken as the negative gradient and it is useful to *restart* the search direction to $-g_c$ every N steps (N being the number of weights in the network). It is easy to check that eqn. 29 requires only $O(N)$ operations for calculating the scalar products $p_c^T g_c$, $p_c^T y_c$, $y_c^T g_c$ and $y_c^T y_c$. Remembering that the search direction has to be used for a fast one-dimensional search (see section 4.1), the total computation per cycle (per epoch) required by the method is a small multiple (2 - 4) of that required by one cycle of gradient descent with a fixed learning rate.

Now, if *exact* line searches are performed, eqn. 29 produces mutually conjugate directions [Shanno 1978]. The difference with other forms of the conjugate gradient method is that the performance of the one-step *positive definite secant update* maintains the "safety" properties even when the search is executed in a small number of one-dimensional trials[22].

While the above method is suitable for batch learning, a proposal for a learning method usable in the on-line procedure has been presented in [le Cun 1989], [Becker and le Cun 1989]. The Hessian is approximated with its diagonal part, so that the matrix-multiplication of the gradient by the inverse Hessian (see Newton's method) is approximated by dividing each gradient component g_n by a *running estimate* of h_{nn} ($\stackrel{\text{def}}{=} \widehat{h}_{nn}$).

At each iteration a particular weight w_n is updated according to the following rule[23]:

$$w_n \leftarrow w_n - \left(\frac{\epsilon}{\mu + \widehat{h}_{nn}}\right) g_n \tag{30}$$

The estimate $\widehat{h}_{nn}$ of the diagonal component of the Hessian is in turn obtained with an exponentially weighted average of the second derivative (or an estimate thereof: $\widehat{\frac{\partial^2 E}{\partial w_n^2}}$), as follows:

$$\widehat{h}_{nn} \leftarrow (1-\gamma)\widehat{h}_{nn} + \gamma\,\widehat{\frac{\partial^2 E}{\partial w_n^2}} \tag{31}$$

Suppose that the weight w_n connects the output of unit j to unit i ($w_n = w_{ij}$, in the double-index notation), a_i is the total input to unit i, $f()$ is the "squashing" function and x_j is the state of unit j. It is easy to derive:

$$\frac{\partial^2 E}{\partial w_{ij}^2} = \frac{\partial^2 E}{\partial a_i^2} x_j^2 \tag{32}$$

---

[22] If the problem is badly scaled, e.g. if the typical magnitude of the variables changes a lot, it is useful to substitute the identity matrix with $H_0 = \max\{|E(w_0)|, \text{typical size of } E\} \cdot D_w^2$, where D_w is a diagonal scaling matrix, such that the new variables $\hat{w} = D_w w$ are in the same range, see [Dennis and Schnabel 1983].

[23] In this rule we omit details related to *weight sharing*, i.e., having more connections controlled by a single parameter.

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

$$\frac{\partial^2 E}{\partial a_i^2} = f'(a_i)^2 \sum_k w_{ki}^2 \frac{\partial^2 E}{\partial a_k^2} - f''(a_i)\frac{\partial E}{\partial x_i} \tag{33}$$

Finally, for the simulations in [le Cun 1989], the term in eqn. 33 with the second derivative of the squashing function is neglected, as in the *Levenberg-Marquardt* method, that will be described in section 7, obtaining

$$\widehat{\frac{\partial^2 E}{\partial a_i^2}} = f'(a_i)^2 \sum_k w_{ki}^2 \frac{\partial^2 E}{\partial a_k^2} \tag{34}$$

Note that a *positive* 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 γ in eqn. 31 are fixed and must be appropriately chosen by the user. The purpose of adding μ to the diagonal approximation is explained by analogy with the trust-region method (see eqn. 22). According to [Becker and le Cun 1989] the method does not bring a tremendous speed-up, but converges reliably without requiring extensive parameter-adjustments.

7 Special Methods for Least Squares

If the error function that is to be minimized is the usual $E = \frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{n_o}(o_{pi} - t_{pi})^2$, learning a set of examples consists in solving a *nonlinear least-squares problem*, for which special methods have been devised. Two of these methods are now described: the first (the *Gauss-Newton method*) is based on simplifying the computation of second derivatives, the second (the *Levenberg-Marquardt method*) is a *trust-region* modification of the former.

Let's define as $R(x)$ the vector[24] whose components are the *residuals* for the different patterns in the training set and output units ($r_{pi}(w) = (o_{pi}(w) - t_{pi})$), so that the error can be expressed as $E = \frac{1}{2}R(w)^T R(w)$. It is straightforward to see that the first and second derivatives of $E(w)$ are, respectively:

$$\nabla E(w) = \sum_{p=1}^{P}\sum_{i=1}^{n_o} r_{pi}(w)\, \nabla r_{pi}(w) = J(w)^T R(w) \tag{35}$$

$$\nabla^2 E(w) = \sum_{p=1}^{P}\sum_{i=1}^{n_o}\left(\nabla r_{pi}(w)\, \nabla r_{pi}(w)^T + r_{pi}(w)\, \nabla^2 r_{pi}(w)\right) = J(w)^T J(w) + S(w) \tag{36}$$

where $J(w)$ is the *Jacobian* matrix $J(x)_{pi,j} = \partial r_{pi}(w)/\partial w_j$ and $S(w)$ is the part of the Hessian containing second derivatives of $r_{pi}(w)$, i.e. $S(w) = \sum_{p=1}^{P}\sum_{i=1}^{n_o} r_{pi}(w)\, \nabla^2 r_{pi}(w)$.

The standard Newton iteration is the following:

$$w_+ = w_c - \left(J(w_c)^T J(w_c) + S(w_c)\right)^{-1} J(w_c)^T R(w_c) \tag{37}$$

The particular feature of the problem is that, while $J(w_c)$ is easily calculated, $S(w_c)$ is not. On the other hand, a secant approximation seems "wasteful" because part of the Hessian (the $J(w_c)^T J(w_c)$

[24] The couples of indices (p, i) can alphabetically ordered, for example, and mapped to a single index.

part) is easily obtained from $J(w)$ and, in addition, the remaining part $S(w)$ is negligible for small values of the residuals. The *Gauss-Newton* method consists in neglecting the $S(w)$ part, so that a single iteration is:

$$w_+ = w_c - (J(w_c)^T J(w_c))^{-1} J(w_c)^T R(w_c) \tag{38}$$

It can be shown that this step is completely equivalent to minimizing the error obtained from using an *affine* model of $R(w)$ around w_c:

$$\min \quad \frac{1}{2}||M_c(w)||^2 \tag{39}$$

$$\text{where} \quad M_c(w) \;=\; R(w_c) \;+\; J(w_c)(w - w_c) \tag{40}$$

The QR factorization method can be used for the solution of eqn.39 (see [Dennis and Schnabel 1983]). The method is locally *q-quadratically* convergent for small residuals. If $J(w_c)$ has full column rank, $J(w_c)^T J(w_c)$ is nonsingular, the Gauss-Newton step is a descent direction and the method can be modified with line searches (*dumped Gauss-Newton method*).

Another modification is based on the *trust-region* idea (see section 4.3) and known as the *Levenberg-Marquardt* method. The step is defined as:

$$w_+ = w_c - (J(w_c)^T J(w_c) \;+\; \mu_c I)^{-1} J(w_c)^T R(w_c) \tag{41}$$

This method can be used also if $J(w)$ does not have full column rank (this happens for example if the number of examples is less than the number of weights).

It is useful to mention that the components of the Jacobian matrix $\partial r_{pi}(w)/\partial w_j$ can be calculated by the usual "chain rule" for derivatives with a number of backpropagation passes equal to the number of output units. If weight w_{ab} connects unit b to unit a (please note that now the usual two-index notation is adopted), one obtains:

$$\frac{\partial r_{pi}(w)}{\partial w_{ab}} = \frac{\partial r_{pi}(w)}{\partial net_{pa}} \frac{\partial net_{pa}}{\partial w_{ab}} = \delta_{pi,a}\, x_{pb} \tag{42}$$

where the term $\delta_{pi,a}$ is defined as $\delta_{pi,a} = (\partial r_{pi}(w)/\partial net_{pa})$ and x_{pb} is the output of unit b. For the output layer, $\delta_{pi,a} = f'(net_{pa})$ if $i = a$, 0 otherwise. For the other layers, $\delta_{pi,a} = f'(net_{pa}) \sum_c \delta_{pi,c} w_{ca}$, summing over the units of the next layer (the one closer to the output).

Software for variants of the *Levenberg-Marquardt* method can be found in [Press et al. 1988]. Other versions are MINPACK and NL2SOL (standard nonlinear least-squares packages)[25]. Additional techniques that are usable for large-scale least squares problems are presented in [Toint 1987]. They are based on adaptive modeling of the objective function and have been used for problems with up to thousands of variables[26]. Additional references are [Gawthrop and Sbarbaro 1990] and [Kollias and Anastassiou 1989]. In [Kollias and Anastassiou 1989] the *Levenberg-Marquardt* technique is combined with the acceleration techniques described in [Jacobs 1988] and [Silva and Almeida 1990].

---

[25] I owe this information to prof. Christopher G. Atkeson. MINPACK is described in [Moré et al. 1980], NL2SOL in [Dennis et al. 1981].

[26] In reality the test problems presented in this paper have a special "partially separable" structure, so that their practical application to multilayer perceptrons with complete connectivity is still a subject of research.

8 Other Heuristic Strategies

Some learning methods have been introduced specifically for backpropagation which show promising performance on some tests problems.

Because the standard algorithm involves selecting appropriate learning and momentum rates, it is convenient to consider ways to *adapt* these parameters during the search process. In this case the trial-and-error selection is avoided and, in addition, the possibility to tune the parameter to the current properties of the "error surface" usually yields faster convergence with respect to using *fixed* coefficients.

An heuristic method for modifying the learning rate is for example described in [Lapedes and Farber 1986], [Vogl et al. 1988] and [Battiti 1989] (the *bold driver* (BD) method). The idea is to increase the learning rate exponentially if successive steps reduce the error, and decrease it rapidly if an "accident" is encountered (increase of the error), until a proper rate is found (if the gradient is significantly different from zero, letting the step go to zero will eventually decrease the error). After starting with a small learning rate[27], its modifications are described by the evolution equation:

$$\epsilon(t) = \begin{cases} \rho\, \epsilon(t-1) & \text{if } E(w(t)) < E(w(t-1)) \\ \sigma^l\, \epsilon(t-1) & \text{if } E(w(t)) > E(w(t-1)) \text{ using } \epsilon(t-1) \end{cases} \tag{43}$$

where ρ is close to one (say $\rho = 1.1$) in order to avoid frequent "accidents" (because the error computation is wasted in these cases), σ is chosen to provide a rapid reduction (say $\sigma = 0.5$), and l is the minimum integer such that the reduced rate ($\sigma^l \epsilon(t-1)$) succeeds in diminishing the error. The performance of this "quick and dirty" version is close and usually better than the one obtained by appropriately choosing a *fixed* learning rate in batch BP.

Suggestions for *adapting* both the search direction and the step along this direction are presented in [Chan and Fallside 1987], [Jacobs 1988]. In [Chan and Fallside 1987] the learning and momentum rates are adapted to the structure of the error surface, by considering the angle θ_k between the last step and the gradient direction and by avoiding "domination" of the weight update by the momentum term (in order to avoid ascent directions). The weight update is:

$$\Delta w_k = \epsilon_k(-g_k + \lambda_k \Delta w_{k-1}) \tag{44}$$

$$\text{where} \quad \lambda_k = \lambda_0 \frac{\|g_k\|}{\|\Delta w_{k-1}\|} \quad ; \quad \epsilon_k = \epsilon_{k-1}\left(1 + \frac{1}{2}\cos\theta_k\right)$$

A comparison of different techniques is presented in [Chan 1990]. In [Jacobs 1988] each individual weight has its own learning rate, that is modified in order to avoid oscillations. In the proposed "delta-bar-delta" method, the learning rate modification is the following:

$$\Delta\epsilon(t) = \begin{cases} \kappa & \text{if } \bar{\delta}(t-1)\delta(t) > 0 \\ -\phi\epsilon(t) & \text{if } \bar{\delta}(t-1)\delta(t) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{45}$$

$$\text{where} \quad \delta(t) = \frac{\partial E(t)}{\partial w(t)} \quad ; \quad \bar{\delta}(t) = (1-\theta)\delta(t) + \theta\bar{\delta}(t-1)$$

The current partial derivative with respect to a weight is compared with an exponentially weighted average $\bar{\delta}(t)$ of the previous derivatives. If the signs agree, the (individual) learning rate ϵ is

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

increased by κ, if they disagree (a symptom of oscillations) it is decreased by a portion ϕ of its value. A similar acceleration technique has been presented in [Silva and Almeida 1990]. An heuristic technique (*quickprop*) "loosely based" on Newton's method is presented in [Fahlman 1988]. In *quickprop* each weight is updated in order to reach the minimum of an independent quadratic model (parabola).

9 Summary and Conclusions

A review of first and second-order methods suitable for learning has been presented. Modifications of first order backpropagation and second-order methods are currently used by many researchers in different fields, both to obtain faster convergence times and to avoid the *meta-optimization* phase (where the user has to select proper parameters for learning, check the numerical stability and, in general, optimize the performance of the learning method). This second point has a considerable impact in order to maintain, and possibly improve, the short development times required by neural networks with respect to methods based on intensive "knowledge engineering".

In the previous sections some of these techniques have been referenced, with particular emphasis on their local and global convergence properties, numerical stability and memory\computation requirements.

Some second-order techniques require a large amount of computation per iteration[28] (of order $O(\mathcal{N}^2)$ or $O(\mathcal{N}^3)$) and\or large amounts of memory (of order $O(\mathcal{N}^2)$). Nonetheless, they are still applicable to problems with a limited number of weights (say < 100) and show superior performance with respect to standard backpropagation, especially if high precision in the input-output mapping function executed by the network is required. For problems with more weights, suitable approximations can be applied, obtaining methods with $O(\mathcal{N})$ behavior, while still maintaining some "safety" and "progress" properties of their parent methods. While the presentation has focussed onto the multilayer perceptron neural network, most of these techniques can be applied to alternative models[29].

It is also worth stressing that problems related to memory requirements are less stringent now than when these methods were invented, and problems related to massive computation can be approached by using *concurrent computation*. Most of the presented techniques are suitable for a parallel implementation, with a speed-up that is approximately proportional to the number of processors employed (see for example [Kramer and Sangiovanni-Vicentelli 1988], [Battiti et al. 1990], [Battiti and Straforini 1991]).

We feel that the cross-fertilization between optimization techniques and neural networks is fruitful and deserves further research efforts. In particular the relevance of second-order techniques to large-scale backpropagation tasks (with thousands of weights and examples) is a subject that deserves additional studies and comparative experiments.

Acknowledgements

The author is indebted to prof. Geoffrey Fox, Roy Williams and Edoardo Amaldi for helpful discussions. Thanks are due to prof. Tommaso Poggio, Christopher Atkeson and Michael Jordan

[28] It is clear that, in order to obtain the total learning time, the estimates must be multiplied by the number of patterns P and the total number of iterations. In this paper these factors are omitted for clarity of exposition.

[29] For example the RBF model introduced in [Broomhead and Lowe 1988], [Poggio and Girosi 1990].

for sharing their views on second-order methods. The results of a "second-order survey" by Eric A. Wan (Neuron Digest 6 - 53) were a useful source of references. The referee's detailed comments are also greatly appreciated.

Part of this work was completed while the author was a research assistant at Caltech. The research group was supported in part by DOE grant DE-FG-03-85ER25009, The National Science Foundation with Grant IST-8700064 and by IBM.

References

[Allred and Kelly 1990] Allred, L.G. and G.E. Kelly. Supervised Learning Techniques for Backpropagation Networks. *Proceedings International Joint Conference on Neural Networks - IJCNN - Washington - USA*, I, 721–728.

[Battiti 1989] Battiti, R. Accelerated Back-Propagation Learning: Two Optimization Methods. *Complex Systems* **3**, 331–342.

[Battiti et al. 1990] Battiti, R., A.M. Colla, L.M. Briano, R. Cecinati and P. Guido. An Application-Oriented Development Environment for Neural Net Models on the Multiprocessor Emma-2. *Proc. IFIP Workshop on Silicon Architectures for Neural Nets - St Paul de Vence - France*, in press.

[Battiti and Masulli 1990] Battiti, R. and F. Masulli. BFGS Optimization for Faster and Automated Supervised Learning. *Proceedings of the International Neural Network Conference (INNC 90) - Paris - France*, 757–760.

[Battiti and Straforini 1991] Battiti, R. and M. Straforini. Parallel Supervised Learning with the Memoryless Quasi-Newton Method. In *Parallel Computing: Problems, Methods and Applications*, P. Messina and A. Murli, eds. . Elsevier, Amsterdam.

[Barnard and Cole 1988] Barnard, E. and R. Cole. A neural-net training program based on conjugate gradient optimization. Oregon Graduate Center, CSE 89–014.

[Becker and le Cun 1989] 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.

[Bengio and Moore 1989] Bengio, Y. and B. Moore. Acceleration of Learning. *Proceedings GNCB-CNR School*, Trento - Italy.

[Bingham 1988] Bingham, J.A.C. *The Theory and Practice of Modem Design*. Wiley, New York, NY.

[Broomhead and Lowe 1988] Broomhead, D.S. and D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems* **2**, 321–355.

[Broyden et al. 1973] Broyden, C.G., J.E. Dennis, and J.J. Moré. On the local and superlinear convergence of quasi-Newton methods. *J.I.M.A.* **12**, 223–246.

[Cardon et al. 1991] Cardon, H., R. van Hoogstraten and P. Davies. A Neural Network Application in Geology: Identification of Genetic Facies. *Proceedings of the International Conference on Artificial Neural Networks (ICANN-91) - Espoo - Finland*, 1519–1522.

[Chan and Fallside 1987] Chan, L.W. and F. Fallside. An adaptive training algorithm for back propagation networks. *Computer Speech and Language* **2**, 205–218.

[Chan 1990] Chan, L. W. Efficacy of different learning algorithms of the backpropagation network. *Proceedings IEEE TENCON-90*.

[Dennis and Schnabel 1983] Dennis, J.E. and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, Englewood Cliffs, NJ.

[Dennis et al. 1981] Dennis, J.E., D.M. Gay and R.E. Welsch. Algorithm 573 NL2SOL-An adaptive nonlinear least-squares algorithm [E4]. TOMS **7**, 369–383.

[Dongarra et al. 1979] Dongarra, J.J., C.B. Moler, J.R. Bunch and G.W. Stewart. LINPACK User's Guide. Siam, Philadelphia.

[Drago and Ridella 1991] Drago, G. P. and S. Ridella. An optimum weights initialization for improving scaling relationships in BP learning. Proceedings of the International Conference on Artificial Neural Networks (ICANN-91) - Espoo - Finland, 1519–1522.

[Fahlman 1988] Fahlman, S. E. An Empirical Study of Learning Speed in Back-Propagation Networks. Preprint CMU-CS-88-162.

[Gawthrop and Sbarbaro 1990] Gawthrop, P. and D. Sbarbaro. Stochastic Approximation and Multilayer Perceptrons: The Gain Backpropagation Algorithm. Complex Systems **4**, 51–74.

[Gill et al. 1981] Gill, P.E., W. Murray and M. H. Wright. Practical Optimization. Academic Press, London.

[Glover 1987] Glover, F. TABU Search Methods in Artificial Intelligence and Operations Research. ORSA Art. Int. Newsletter, Vol. 1, No 2.6.

[Goldfarb 1976] Goldfarb, D. Factorized variable metric methods for unconstrained optimization. Math. Comp. **30**, 796–811.

[Goldstein 1967] Goldstein, A.A. Constructive Real Analysis. Harper and Row, New York, NY.

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

[Johansson et al. 1990] Johansson, E.M., F.U. Dowla, D.M. Goodman. Backpropagation Learning for Multi-Layer Feed-Forward Neural Networks Using the Conjugate Gradient Method. Lawrence Livermore National Laboratory, Preprint UCRL-JC-104850.

[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-Vicentelli 1988] Kramer, A. H. and A. Sangiovanni-Vicentelli. Efficient Parallel Learning Algorithms For Neural Networks. In Advances in Neural Information Processing Systems, Vol. **1**, 75–89. Morgan Kaufmann, San Mateo, CA.

[Lapedes and Farber 1986] Lapedes, A. and R. Farber. A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition. Physica **22 D**, 247–259.

[le Cun 1986] le Cun, Y. HLM: A Multilayer Learning Network. Proceedings of the 1986 Connectionist Models Summer School - Pittsburgh, 169–177.

[le Cun 1989] le Cun, Y. Generalization and Network Design Strategies. In Connectionism in Perspective, 143–155. North Holland, Amsterdam.

[le Cun et al. 1991] le Cun, Y., I. Kanter and S.A. Solla. Second Order Properties of Error Surfaces: Learning Time and Generalization. In Neural Information Processing Systems - NIPS 3, 918–924. Morgan Kaufmann, San Mateo, CA.

[Luenberger 1973] Luenberger, D. G. Introduction to linear and nonlinear programming. Addison Wesley, New York, NY.

[Luo 1991] Luo, Zhi-Quan. On the Convergence of the LMS algorithm with Adaptive Learning Rate for Linear Feedforward Networks. Neural Computation **3**, 227–245.

[Malferrari et al. 1990] Malferrari, L., R. Serra and G. Valastro. Using Neural Networks for Signal Analysis in Oil Well Drilling. Proceedings of the III Italian Workshop on Parallel Architectures and Neural Networks - Vietri s/m Salerno - Italy, 345–353. World Scientific, Singapore.

[Moller 1990] Moller, M.F. A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. PB-339 Preprint. Computer Science Department, University of Aarhus, Denmark.

[Moré et al. 1980] Moré, J.J., B.S. Garbow and K.E. Hillstrom. User guide for MINPACK-1. Argonne National Labs Report ANL-80-74.

[Nocedal 1980] Nocedal, J. Updating quasi-Newton matrices with limited storage. Mathematics of Computation **35**, 773–782.

[Parker 1987] 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, II-593 – II-600.

[Poggio and Girosi 1990] Poggio, T. and F. Girosi. Regularization Algorithms for Learning that are Equivalent to Multilayer Networks. Science **247**, 978–982.

[Press et al. 1988] Press, W.H., B.P. Flannery, S.A. Teukolsky and W.T. Wetterling. Numerical Recipes in C. Cambridge University Press, Cambridge, MA.

[Rumelhart et al. 1986] Rumelhart, D.E. and J. L. McClelland (eds.),Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations. MIT Press, Cambridge, MA.

[Sejnowski and Rosenberg 1986] Sejnowski, T.J. and C.R. Rosenberg. NETtalk: a parallel network that learns to read aloud. The Johns Hopkins University EE and CS Technical Report JHU/EECS-86/01.

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

[Silva and Almeida 1990] Silva, F. and L. Almeida. Acceleration Techniques for the Backpropagation Algorithm. Lecture Notes in Computer Science **412**, 110–119. Springer Verlag, Berlin.

[Toint 1987] Toint, L. On Large Scale Nonlinear Least Squares Calculations. SIAM J. Sci. Stat. Comput. **8 (3)**, 416–435.

[Vincent 1991] Vincent, J.M. Facial Feature Location in Coarse Resolution Images by Multi-Layerd Perceptrons. Proceedings of the International Conference on Artificial Neural Networks (ICANN-91) - Espoo - Finland, 821–826. North Holland, Amsterdam.

[Vogl et al. 1988] Vogl,T.P., J.K. Mangis, A.K. Rigler, W. T. Zink and D. L. Alkon. Accelerating the Convergence of the Back-Propagation Method. Biological Cybernetics **59**, 257–263.

[Watrous 1987] Watrous, R. Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization. Technical Report MS-CIS-87-51, Univ. of Penn.

[Webb et al. 1988] Webb, A. R., D. Lowe and M. D. Bedworth. A Comparison of Non-linear Optimization Strategies for Adaptive Feed-forward Layered Networks. RSRE MEMO 4157, Royal Signals and Radar Establishment, Malvern, England.

[White 1989] White, H. Learning in Artificial Neural Networks: a Statistical Perspective. Neural Computation **1**, 425–464.

[Widrow and Stearns 1985] Widrow, B. and S.D. Stearns. Adaptive Signal Processing. Prentice Hall, Englewood Cliffs, NJ.