A Marquardt algorithm for choosing the step-size in backpropagation learning with conjugate gradients

Peter M. Williams
School of Cognitive and Computing Sciences
University of Sussex

13 February 1991

Standard learning in feed-forward networks uses simple gradient descent, sometimes with a "momentum" term. Gradient descent is very inefficient. The momentum method is an improvement though it is ad hoc and shares with the steepest descent method the disadvantage of requiring an arbitrary choice of parameter.

Let $E(\mathbf{w})$ be the error function of the network where $\mathbf{w}$ is the weight vector (all the weights and "biases" in the network in some order or other). The aim is assumed to be that of unconstrained minimisation of E.

Suppose that $\mathbf{w}_n$ is the weight vector at the n'th iteration. The next iteration involves choosing a suitable non-zero *search direction* $\mathbf{s}_n$ and *step length* α_n. The weight vector is then updated by the rule

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \alpha_n \mathbf{s}_n. \tag{1}$$

Simple gradient descent uses

$$\mathbf{s}_n = -\mathbf{g}_n$$

where $\mathbf{g}_n = \nabla E(\mathbf{w}_n)$ with a fixed step size $\alpha_n = \alpha$.

The momentum term method also uses a fixed step size α but chooses, as search direction, a linear combination of the current direction of steepest descent and the preceding search direction

$$\mathbf{s}_n = -\mathbf{g}_n + \beta \mathbf{s}_{n-1}$$

where β is some fixed constant. The algorithm is started in the direction of steepest descent: $\mathbf{s}_0 = -\mathbf{g}_0$.

The strategy of choosing a linear combination of $-\mathbf{g}_n$ and $\mathbf{s}_{n-1}$ is a good one. The disadvantage of the method, however, is that both α and β remain fixed throughout and there is no theoretical basis for choosing their values. Standard optimisation techniques, however, can be used to remedy this by varying α and β adaptively at each iteration.

Choosing the search direction

Many methods for unconstrained minimization are based on *quadratic models* of the objective function. These work well, or even exactly, for quadratic functions with positive

definite Hessian. Nonetheless they can be applied to minimize general functions.[1] Minimization of a quadratic function of N variables can be achieved in at most N steps if an exact line search is used and successive search directions are *conjugate* with respect to the Hessian.[2] Assuming the update rule

$$\mathbf{s}_{n+1} = -\mathbf{g}_{n+1} + \beta_n \mathbf{s}_n$$

the simplest prescription for conjugate gradient directions is the **Fletcher-Reeves** formula

$$\beta_n = \frac{\mathbf{g}_{n+1} \cdot \mathbf{g}_{n+1}}{\mathbf{g}_n \cdot \mathbf{g}_n}.$$

There are several alternatives for β_n that are equivalent for quadratic functions with exact line search. The **Polak-Ribière** formula, for instance, employs

$$\beta_n = \frac{(\mathbf{g}_{n+1} - \mathbf{g}_n) \cdot \mathbf{g}_{n+1}}{\mathbf{g}_n \cdot \mathbf{g}_n}$$

where $\mathbf{g}_n \cdot \mathbf{g}_{n+1}$ vanishes for exact line searches but can be helpful when searching is done only approximately. In particular, if the algorithm is making poor progress, then $\mathbf{g}_{n+1} \approx \mathbf{g}_n$ so that $\beta_n \approx 0$ and the search restarts close to the direction of steepest descent.

Another possibility is to replace the denominator $\mathbf{g}_n \cdot \mathbf{g}_n$ in the Polak-Ribière formula by $-\mathbf{s}_n \cdot \mathbf{g}_n$. This is also equivalent to the Fletcher-Reeves formula, assuming exact line searches, but has advantages for non-quadratic functions. This gives the **Hestenes-Stiefel** formula

$$\beta_n = \frac{(\mathbf{g}_n - \mathbf{g}_{n+1}) \cdot \mathbf{g}_{n+1}}{\mathbf{s}_n \cdot \mathbf{g}_n}.$$

which will be used in the algorithm presented below.

Choosing the step size

Assuming a current weight vector $\mathbf{w}_n$ and a search vector $\mathbf{s}_n$, the ideal is to choose α_n so as to minimise

$$f(\alpha) = E(\mathbf{w}_n + \alpha \mathbf{s}_n)$$

with respect to α. The weight vector is then updated by the rule (1). This reduces the problem to a single dimension.

An exact line search involving several, possibly many, function evaluations is computationally expensive. An alternative is to fit a quadratic $\hat{f}$ to $f(0)$, $f'(0)$, $f''(0)$ and to choose α_n as the minimum of $\hat{f}$ giving

$$\alpha_n = -\frac{f'(0)}{f''(0)}. \tag{2}$$

$f'(0)$ is the directional derivative of $E(\mathbf{w}_n)$ along $\mathbf{s}_n$ calculated by

$$f'(0) = \mathbf{s}_n \cdot \nabla E(\mathbf{w}_n). \tag{3}$$

[1] See [2, p.24] for a more general discussion of quadratic models.

[2] For details of the method of conjugate gradients see [2, Ch.4], [3, Sec.4.8.3] and[4, passim].

Estimating the curvature $f''(0)$ by differencing first derivatives gives

$$f''(0) \;\approx\; \mathbf{s}_n \cdot \frac{\nabla E(\mathbf{w}_n + \sigma \mathbf{s}_n) - \nabla E(\mathbf{w}_n)}{\sigma} \tag{4}$$

for sufficiently small σ. Choice of σ is discussed in the next section.

Unfortunately $f(\alpha)$ may be only poorly approximated by a quadratic for $\alpha > 0$. Furthermore, assuming that $f'(0) < 0$, the method breaks down altogether, as a method for finding a *minimum* of $f(\alpha)$, unless $f''(0) > 0$. One remedy is to use a *restricted-step method* in which the step size is restricted by the region of validity of the second-order Taylor series expansion. Møller [5] proposes adding a positive scalar multiple of the identity $\lambda_n I$ to the Hessian as in Levenberg-Marquardt algorithms.[3] In general the parameter λ_n is chosen sufficiently large to make the Hessian positive definite. For the present one-dimensional approach, λ_n is chosen to make the curvature $f''(0)$ positive. The step size α_n is then given by

$$\alpha_n = -\frac{f'(0)}{f''(0) + \lambda_n \|\mathbf{s}_n\|^2} \qquad (\|\mathbf{s}_n\|^2 = \mathbf{s}_n \cdot \mathbf{s}_n) \tag{5}$$

where λ_n is assumed to have been chosen so that the denominator is strictly positive.

λ_n can be changed adaptively to reflect how well f is approximated by the current quadratic model

$$\hat{f}(\alpha) = f(0) + \alpha f'(0) + \tfrac{1}{2}\alpha^2 \left(f''(0) + \lambda_n \|\mathbf{s}_n\|^2\right).$$

If α_n is the step-size used on the n'th iteration, the *actual reduction* in error is

$$\Delta f = f(0) - f(\alpha_n)$$

while the *predicted reduction* according to the quadratic model is

$$\Delta \hat{f} = \hat{f}(0) - \hat{f}(\alpha_n).$$

The ratio

$$\rho_n = \frac{\Delta f}{\Delta \hat{f}} = 2\,\frac{f(\alpha_n) - f(0)}{\alpha_n f'(0)}$$

provides a measure of accuracy of the current approximation. Ideally $\rho_n = 1$. Following Fletcher [2, p.96], λ_n can be adapted by the rule

if $\rho_n < 0.25$, set $\lambda_{n+1} = 4\lambda_n$

if $\rho_n > 0.75$, set $\lambda_{n+1} = \lambda_n/2$

otherwise, set $\lambda_{n+1} = \lambda_n$.

Thus λ_n is increased if the approximation is poor and decreased if it is good. Notice that, according to (5), an increase in λ_n means a decrease in the step size or "trust-region". A decrease in λ_n, when the approximation is good, means an increase in the trust-region.[4]

Successively increasing λ_n by a multiplicative factor can also be used to ensure that the modified curvature $f''(0) + \lambda_n \|\mathbf{s}_n\|^2$ is positive. However this can be done in a single step without further gradient evaluations. A simple rule is

[3] See [2, Ch.5] and [3, p.113] for discussion of restricted-step or "trust-region" methods.

[4] Fletcher [1] suggests a more sophisticated adaptation of λ_n in the case where $\rho_n < 0.25$. The multiplicative factor is then chosen in the range [2,10] on the basis the location of the minimum of a quadratic fitted to $f(0)$, $f'(0)$ and $f(\alpha)$. This factor is then $2 - \rho_n$ provided it lies in the interval [2,10], otherwise the nearest endpoint. However, this appears to have only a marginal effect. An alternative and improved rule is given in step 6 of the algorithm presented below.

if $f''(0) + \lambda_n^{(\text{old})} \|\mathbf{s}_n\|^2 \leq 0$, reset

$$\lambda_n^{(\text{new})} = \lambda_n^{(\text{old})} - \frac{f''(0)}{\|\mathbf{s}_n\|^2}$$

following which

$$f''(0) + \lambda_n^{(\text{new})} \|\mathbf{s}_n\|^2 = \lambda_n^{(\text{old})} \|\mathbf{s}_n\|^2 > 0.$$

Approximating the second derivative

The simplest way of determining σ in (4) is to choose a small constant $\epsilon > 0$ and set

$$\sigma_n = \frac{\epsilon}{\|\mathbf{s}_n\|}$$

on the n'th iteration. The renormalization ensures uniform scaling for varying directions and gradients. If f were in fact quadratic the exact choice of σ would be unimportant and there would be no need for small ϵ. Figure 1, however, shows that there is no particular advantage in having $\sigma_n \ll 1$ even when f is not quadratic. When $\lambda_n \approx 0$ the method proposed by (2), (3), (4) is equivalent to fitting a straight line to $f'(0)$ and $f'(\sigma_n)$ to give α_n as the estimated zero crossing of f'. The ideal σ_n would be one for which $\sigma_n = \alpha_n$. A

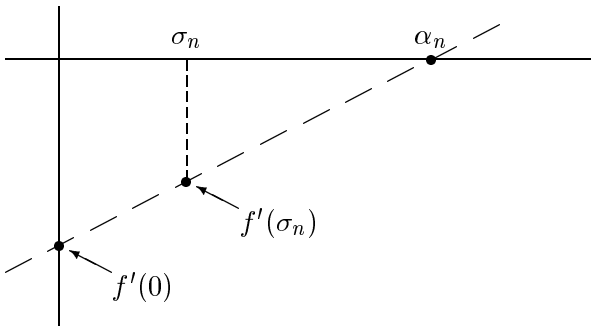

Figure 1: The extrapolation used to determine α_n.

rough equality between σ_n and α_n is also appropriate for $\lambda > 0$. This suggests a way of adapting ϵ and hence σ_n from cycle to cycle.

Let ϵ_0 be chosen initially, say $\epsilon_0 = 10^{-3}$, and let π be some number in the range $0 \leq \pi \leq 1$. Then ϵ_n is adapted by the rule

$$\epsilon_{n+1} = \epsilon_n \left(\frac{\alpha_n}{\sigma_n}\right)^{\pi}$$

where σ_n is given on the n'th iteration by

$$\sigma_n = \frac{\epsilon_n}{\|\mathbf{s}_n\|}.$$

Choosing $\pi = 0$ is equivalent to using the initial value ϵ_0 throughout. Non-zero values of π adapt ϵ_n so that, on average, the values of α_n and σ_n tend to be equalized. $\pi = 0.05$ or $\pi = 0.1$ have been found satisfactory. Although this provides only a marginal improvement over an optimally chosen but fixed value of ϵ, it provides a stable and automatic way of choosing a suitable ϵ and hence σ_n.

The algorithm

The algorithm can be stated as follows:

0. choose weight vector $\mathbf{w}_0$, scalars $\epsilon_0 > 0$, $\lambda_0 > 0$, $\pi \geq 0$ and initialize search direction:

$$\mathbf{g}_0 = \nabla E(\mathbf{w}_0)$$

$$\mathbf{s}_0 = -\mathbf{g}_0$$

$$\text{success} = \text{true},\ S = 0,\ n = 0$$

1. if success = true calculate first and second order directional derivatives:

$$\mu_n = \mathbf{s}_n \cdot \mathbf{g}_n \qquad \text{(directional gradient)}$$

$$\text{if } \mu_n \geq 0, \text{ set } \mathbf{s}_n = -\mathbf{g}_n,\ \mu_n = \mathbf{s}_n \cdot \mathbf{g}_n,\ S = 0$$

$$\kappa_n = \mathbf{s}_n \cdot \mathbf{s}_n, \quad \sigma_n = \frac{\epsilon_n}{\sqrt{\kappa_n}}$$

$$\gamma_n = \mathbf{s}_n \cdot \frac{\nabla E(\mathbf{w}_n + \sigma_n \mathbf{s}_n) - \nabla E(\mathbf{w}_n)}{\sigma_n} \qquad \text{(directional curvature)}$$

2. increase the working curvature: $\delta_n = \gamma_n + \lambda_n \kappa_n$
3. if $\delta_n \leq 0$ make δ_n positive and increase λ_n:

$$\delta_n = \lambda_n \kappa_n$$

$$\lambda_n = \lambda_n - \frac{\gamma_n}{\kappa_n}$$

4. calculate step size and adapt ϵ:

$$\alpha_n = -\frac{\mu_n}{\delta_n}$$

$$\epsilon_{n+1} = \epsilon_n \left(\frac{\alpha_n}{\sigma_n}\right)^{\pi}$$

5. calculate the comparison ratio:

$$\rho_n = \frac{2[E(\mathbf{w}_n + \alpha_n \mathbf{s}_n) - E(\mathbf{w}_n)]}{\alpha_n \mu_n}$$

$$\text{success} = \rho_n \geq 0$$

6. if $\rho_n < 0.25$, set $\lambda_{n+1} = \min\{\lambda_n + \frac{\delta_n(1-\rho_n)}{\kappa_n},\ \lambda_{\max}\}$

   if $\rho_n > 0.75$, set $\lambda_{n+1} = \max\{\lambda_n/2,\ \lambda_{\min}\}$

   otherwise, set $\lambda_{n+1} = \lambda_n$

7. if success = true then adjust weights:

   $$\mathbf{w}_{n+1} = \mathbf{w}_n + \alpha_n \mathbf{s}_n$$

   $$\mathbf{g}_{n+1} = \nabla E(\mathbf{w}_{n+1})$$

   $$S = S + 1$$

   else leave weights unchanged:

   $$\mathbf{w}_{n+1} = \mathbf{w}_n$$

   $$\mathbf{g}_{n+1} = \mathbf{g}_n$$

8. choose new search direction:

   if $S = S_{\max}$ restart algorithm in direction of steepest descent:

   $$\mathbf{s}_{n+1} = -\mathbf{g}_{n+1}$$

   success = true, $S = 0$

   else

   if success = true create new conjugate direction:

   $$\beta_n = \frac{(\mathbf{g}_n - \mathbf{g}_{n+1}) \cdot \mathbf{g}_{n+1}}{\mu_n}$$
   $$\mathbf{s}_{n+1} = -\mathbf{g}_{n+1} + \beta_n \mathbf{s}_n$$

   else use current direction again:

   $$\mathbf{s}_{n+1} = \mathbf{s}_n$$
   $$\mu_{n+1} = \mu_n,\ \kappa_{n+1} = \kappa_n,\ \sigma_{n+1} = \sigma_n,\ \gamma_{n+1} = \gamma_n$$

9. if $\|\mathbf{g}_{n+1}\| < \tau$ return $\mathbf{w}_{n+1}$ as desired minimum, else go to 1 with $n = n + 1$.

Notes on the algorithm

0. $\epsilon_0 = 10^{-3}$ is satisfactory though not critical if $\pi > 0$. If a non-zero value of π is chosen then $\pi = 0.05$ or $\pi = 0.1$ are recommended. The initial value of λ is not critical though $\lambda_0 = 1$ is a natural choice. The algorithm starts in the direction of steepest descent.

1. Apart from the initial cycle, this step is only executed if the last cycle succeeded in error reduction. Otherwise no change in the weight vector has been made and this information is already known. Neither the Hestenes-Stiefel formula nor the Polak-Ribière formula guarantees that $\mathbf{s}_n$ is a *descent* direction, though usually it is. If $\mu_n \geq 0$, a restart is made in the direction of steepest descent for which $\mu_n = -\mathbf{g}_n \cdot \mathbf{g}_n$ is negative, otherwise the algorithm would have terminated at the last step of the previous cycle, assuming the two-norm is used.

3. After this step, $\delta_n = \gamma_n + \lambda_n \kappa_n$ as before, but with the new value of λ_n.

5. Remember that $\mu_n < 0$. The choice of $\rho_n \geq 0$ rather than $\rho_n > 0$ is deliberate. It safeguards against the algorithm getting stuck owing to limited floating-point precision. An alternative is to restart in the direction of steepest descent after a given number, 10 say, of consecutive failures.

6. λ_n must stay in the range $0 < \lambda_n < \infty$, otherwise no further rescaling is possible. $\lambda_{\min}$ and $\lambda_{\max}$ can be of the order of the smallest and largest positive floating point numbers provided by the implementation. When $\rho_n < 0.25$, the proposed rule increases λ_n by more than a factor 4, the intention being to avoid the possibility of more than 2 or 3 successive failures.

8. S is the total number of successes since the last restart in the direction of steepest descent. By default $S_{\max}$ is the dimension of the weight vector (the total number of weights and biases). For large scale problems more frequent restarts may be advisable. Powell [6] suggests setting $\beta_n = 0$ when $\beta_n < 0$ when using the Polak-Ribière formula. This is not recommended for the present algorithm.

9. For the two-norm $\|\mathbf{g}_{n+1}\| = \sqrt{\mathbf{g}_{n+1} \cdot \mathbf{g}_{n+1}}$. Gill et al (1981, p.307), however, recommend using the infinity-norm if the number of variables is large. The choice of τ is up to the user.

Notes on complexity

Time The algorithm requires both function values $E(\mathbf{w})$ and gradients $\nabla E(\mathbf{w})$. For feed-forward networks a function evaluation requires a forward pass for each pattern in the batch. A gradient evaluation requires both a forward and a backward pass for each pattern. In fact a gradient evaluation provides the function value at no significant extra cost. On the other hand, if we first calculate $E(\mathbf{w})$ and then subsequently calculate $\nabla E(\mathbf{w})$, the former work will have to be redone unless the output of each unit for each pattern in the batch has been stored, which is often impractical.

Each cycle of the algorithm involves at most two gradient evaluations, assuming these also give the function value. Suppose that initially, or after a previous cycle, both $E(\mathbf{w}_n)$ and $\nabla E(\mathbf{w}_n)$ are known at the beginning of step 1. Step 1 requires an evaluation of $\nabla E(\mathbf{w}_n + \sigma_n \mathbf{s}_n)$, depending on whether or not success = true. Step 5 requires a single

function evaluation $E(\mathbf{w}_n+\alpha_n\mathbf{s}_n)$ though it is worth performing the full gradient evaluation $\nabla E(\mathbf{w}_n+\alpha_n\mathbf{s}_n)$ at this stage. If an error reduction results, $\mathbf{w}_n+\alpha_n\mathbf{s}_n$ will become the new weight vector $\mathbf{w}_{n+1}$ in step 7 and $\nabla E(\mathbf{w}_{n+1})$ will then already be known. If no error reduction occurs, the extra computation will have been wasted. On the other hand, if an error reduction does occur, the work involved in calculating only the function value $E(\mathbf{w}_n+\alpha_n\mathbf{s}_n)$ in step 5 will have to be redone. Assuming that successes are more common than failures, it is better on average to calculate the gradient in step 5. Note that at the end of the cycle both $E(\mathbf{w}_{n+1})$ and $\nabla E(\mathbf{w}_{n+1})$ are known.

All other significant calculations in a cycle are inner products. Each requires N multiplications and additions, where N is the number of weights. This is comparable to a forward pass of a single pattern. If $P \gg 1$, where P is the number of patterns in a batch, the cost of the inner product calculations is not significant.

Space Memory must be allocated for storing "current" and "alternative" weight and gradient arrays. "Alternative" refers to $\mathbf{w}_n+\sigma_n\mathbf{s}_n$ in step 1 and to $\mathbf{w}_n+\alpha_n\mathbf{s}_n$ in step 5. The first need not stored beyond step 1. If a successful error reduction is made, "alternative" is made "current" at a suitable stage in steps 7 and 8.

Storage is also required for the current search direction, on top of whatever is required for implementing back-propagation.

The scalars $\mu, \kappa, \sigma, \gamma$ need to be stored between cycles since, if success = true, they will not be recalculated in step 1.

References

[1] R. Fletcher. A modified Marquardt subroutine for nonlinear least squares. Report R6799, Atomic Energy Research Establishment, Harwell, England, 1971.

[2] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, second edition, 1987.

[3] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, 1981.

[4] Magnus R. Hestenes. *Conjugate Direction Methods in Optimization.* Springer-Verlag, 1980.

[5] Martin F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. Report DAIMI PB-339, Aarhus University, July 1990.

[6] M. J. D. Powell. Nonconvex minimization calculations and the conjugate gradient method. In D. F. Griffiths, editor, *Numerical Analysis Proceedings, Dundee 1983*, Berlin, 1984. Springer Verlag.