Notes on NLCG design and implementation

Frederik Eaton

31 May, 2025

The optimization procedure is the heart of any machine learning application and deserves special consideration. The NLCG optimization algorithm is efficient in both time and memory and seems difficult to improve upon, although at the same time hard to understand. To make the code more understandable and maintainable it seemed wise to include a short summary and justification of the NLCG algorithm. Our primary reference is Shewchuk, 1994, "An Introduction to the Conjugate Gradient Method ...". Any page numbers in this document are referring to that paper.

1 The problem setup

In the functional form of CG and NLCG we are minimizing (p. 2) the quadratic form

$$f(x) = \frac{1}{2}x^\top Ax - b^\top x + c \tag{1}$$

The minus sign is from Shewchuk and relates to the matrix form of the minimization problem,

$$Ax = b \tag{2}$$

The CG method calculates $x = A^{-1}b$. The NLCG method applies to a more general objective function f, which can be arbitrary if smooth. In this case A will be the Hessian $\frac{\partial^2 f}{\partial x^2}(x)$ and b the negative gradient at 0: $b = -f'(0)$.

It seems most sensible to motivate the NLCG optimization procedure using some simple derivations from calculus, although the language in Shewchuk uses more linear algebra.[1]

2 Derivations

1. **Whether stepping in a given direction will take us downhill**

$$\left.\frac{\mathrm{d}}{\mathrm{d}\alpha}f(x + \alpha d)\right|_{\alpha=0} = f'(x)^\top d \tag{3}$$

[1]For example the negative gradient is called the "residual" r.

If the inner product of d with the gradient is negative, then a small motion in the direction of d will take us downhill.

2. **The optimal step distance to take in any given direction**

   From derivation 1, after taking an optimal step in direction d, the inner product of d with the new gradient will be 0:

$$
\begin{aligned}
0 &= f'(x+\alpha)^\top d && (4)\\
&= (A(x+\alpha d)-b)^\top d && (5)\\
&= (f'(x)+\alpha A d)^\top d && (6)\\
\implies \alpha &= -\frac{f'(x)^\top d}{d^\top A d} && (7)
\end{aligned}
$$

   Here Ad can be calculated as an HVP at little cost.[2] This is equivalent to Newton's method in 1 dimension. The denominator could be written $\frac{\mathrm{d}^2}{\mathrm{d}\alpha^2} f(x+\alpha d)\Big|_{\alpha=0}$ just as the numerator is $\frac{\mathrm{d}}{\mathrm{d}\alpha} f(x+\alpha d)\big|_{\alpha=0}$.

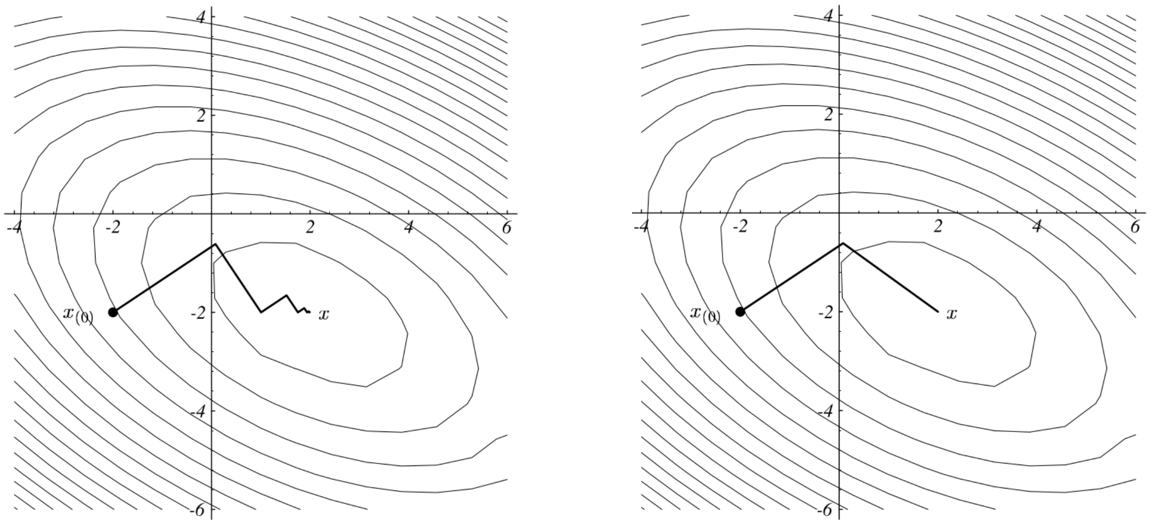

Figure 1: Steepest descent (left) versus Conjugate gradients (right). Copied from p. 8 and p. 32 of Shewchuk.

[2]It seems that Shewchuk does not know about how to calculate the HVP without the Hessian, see his arguments on p. 46 for example; in fact his entire 1994 article never mentions backpropagation either, which had been well described by the mid-1980s.

3. **How to avoid undoing our progress when we take the next step**

   Consider $g(\alpha, \beta) = f(x + \alpha d + \beta e)$, and let $\beta^*(\alpha) = \min_\beta g(\beta, \alpha)$. What we want is to choose direction vectors d and e such that the optimal *beta* does not depend on our choice of α: $\frac{\mathrm{d}}{\mathrm{d}\alpha}\beta^* = 0$. By the implicit function theorem, we have

$$0 = \frac{\mathrm{d}}{\mathrm{d}\alpha}\beta^* = (\frac{\partial^2 g}{\partial \beta^2})^{-1}(\frac{\partial^2 g}{\partial\alpha\partial\beta}) \tag{8}$$

$$= (e^\top Ae)^{-1}(d^\top Ae) \tag{9}$$

$$\implies d^\top Ae = 0 \tag{10}$$

   This is to say that vectors d and e are conjugate relative to A, or we sometimes say A-perpendicular or A-orthogonal. Thus if we choose successive direction vectors to be conjugate to all previous ones (relative to A), then we can avoid backtracking. Note that the order of steps does not matter when the directions are conjugate, as each step length remains optimal independently of the others. In particular the gradient condition in derivation 1 will hold for all previous directions, implying that the new gradient will be perpendicular to each of them.

4. **How to pick a new direction vector**

   At each step of our optimization, we want as we have seen to choose a search direction that is conjugate or A-perpendicular to all previous directions. The natural quantity from which to derive the new direction is the gradient $-r_n$, which is perpendicular to all previous directions $d_j : j \in [1, n-1]$. We can use HVPs to project r_n onto the subspace which is A-perpendicular to d_{n-1}, and propose this as the next search direction d_n:

$$d_n = r_n + \beta_n d_{n-1} \tag{11}$$

$$d_1 = r_1 \tag{12}$$

   Then we can solve[3] for β_n:

$$d_n A d_{n-1} = 0 \tag{13}$$

$$\implies r_n A d_{n-1} + \beta_n d_{n-1} A d_{n-1} = 0 \tag{14}$$

$$\implies \beta_n = -\frac{r_n A d_{n-1}}{d_{n-1} A d_{n-1}} \tag{15}$$

   With this choice of d_n, we can show that conjugacy is obtained with respect to all previous search directions as well. This is because the new subspace D_n, spanned by

[3]The expression on p. 32 of Shewchuk is different, it is given as $\beta_n = \frac{r_n^\top r_n}{r_{n-1}^\top r_{n-1}}$, which as we show later is equivalent to ours.

the search directions up to d_n, contains a copy of AD_{n-1}, implying that r_{n+1}, which is perpendicular to D_n, will be A-perpendicular to D_{n-1}. More precisely, for the step $x_{n+1} = x_n + \alpha_n d_n$, we have the following recurrence for the negative gradient:

$$r_{n+1} = r_n - \alpha A d_n \tag{16}$$

$$\implies A d_n = \frac{1}{\alpha}(r_n - r_{n+1}) \tag{17}$$

But if the next direction vector is always projected from r, then $r_n = d_n - \beta_n d_{n-1}$ from above, so

$$A d_n = \frac{1}{\alpha}(d_n - \beta_n d_{n-1} - (d_{n+1} - \beta_{n+1} d_n)) \tag{18}$$

We have written Ad_n as a linear combination of d_{n+1}, d_n, and d_{n-1}. Since, from the previous derivations, $r_n \perp d_j$ for $j \leq n-1$, equation 18 implies that for $j \leq n-2$,

$$r_n \perp A d_j \implies r_n A d_j = 0 \tag{19}$$

This says r_n is A-perpendicular to d_{n-2} and all earlier directions. Assuming by induction that d_{n-1} has this same property, i.e. $d_{n-1} A D_{n-2} = 0$, we find that when d_n is formed from a linear combination of r_n and d_{n-1} so as to be A-orthogonal to d_{n-1}, it will also be A-orthogonal to all previous d's, $d_1, \ldots d_{n-2}$, as both r_n and d_{n-1} have this (linear) A-orthogonality property with the subspace D_{n-2}.

3 The algorithm and its variations

From the foregoing observations we can write down the steps of the CG or NLCG algorithm, given an input f which is approximately quadratic, and a starting point x_1.

$$r_1 = -f'(x_1) \tag{20}$$

$$d_1 = r_1 \tag{21}$$

$$x_n = x_{n-1} + \alpha_{n-1} d_{n-1} \tag{22}$$

$$r_n = -f'(x_n) \tag{23}$$

$$d_n = r_n + \beta_n d_{n-1} \tag{24}$$

where

$$\alpha_n = -\frac{f'(x_n)^\top d_n}{d_n^\top A d_n} \quad \text{from derivation 2} \tag{25}$$

$$\beta_n = -\frac{r_n A d_{n-1}}{d_{n-1} A d_{n-1}} \quad \text{from derivation 4} \tag{26}$$

$$\tag{27}$$

We can calculate all of these quantities efficiently if we keep track of the HVP Ad_n at each step. However, it is possible to use the orthogonality properties of d_j and r_j to rewrite some of the HVPs, following Shewchuk, from 16:

$$\beta_n = -\frac{r_n^\top A d_{n-1}}{d_{n-1}^\top A d_{n-1}} \tag{28}$$

$$= -\frac{r_n^\top (r_{n-1} - r_n)}{d_{n-1}^\top (r_{n-1} - r_n)} \tag{29}$$

But $r_n^\top r_{n-1} = r_n^\top (d_{n-1} - \beta_{n-1} d_{n-2}) = 0$ and $d_n^\top r_n = (r_n + \beta_n d_{n-1})^\top r_n = r_n^\top r_n$, so

$$\beta_n = \frac{r_n^\top r_n}{r_{n-1}^\top r_{n-1}} \tag{30}$$

is equivalent and has no HVPs in it. The expression Shewchuk uses for α_n, however, includes an HVP in the denominator because the new gradient r_{n+1} is not available at this stage of the computation. Only the numerator is rewritten:

$$\alpha_n = \frac{r_n^\top d_n}{d_n^\top A d_n} \tag{31}$$

$$= \frac{r_n^\top r_n}{d_n^\top A d_n} \tag{32}$$

Shewchuk advocates recalculating the gradient at each step of the NLCG algorithm, but for the CG algorithm he uses the recurrence 16:

$$r_{n+1} = r_n - \alpha_n A d_n$$

With these modifications, we have the CG algorithm from p. 32 of Shewchuk. (XXX NLCG differences)

We are not sure which of these alterations to the "naive" formulation of CG is necessary or useful in the context of non-linear optimization. For example the HVP Ad_n can be computed together with the gradient $-r_n = f'(x_n)$, so it is not more efficient to use a recurrence for r_n unless we can isolate the dual part of a computation (`tape_get_dual`) and compute only the HVP. In fact the NLCG algorithm on p. 42 specifies recomputing r from the negative gradient at each step

$$r_n = -f'(x_n) \tag{33}$$

The choice of step length α can be as above in the CG algorithm, although Shewchuk advocates a more general line search to reliably minimize $f(x + \alpha d)$ with respect to α. This could also involve additional Newton steps on α, or the secant method (p. 52 and 53). Reliably minimizing α helps guarantee that the new gradient is orthogonal to the last search direction, so that the assumptions of the algorithm continue to hold, but on the other hand if we are adjusting α to step out of a concavity then the old Hessian A is unlikely to describe the curvature at the new x in the first place. [4]

The projection coefficient β_n is often calculated as

$$\beta_n^{\mathrm{PR}} = \frac{r_n^\top (r_n - r_{n-1})}{r_{n-1}^\top r_{n-1}} \tag{34}$$

which is called the Polak-Ribiere formula. (Equation 30 is called the Fletcher-Reeves formula)

This choice of β gives us "restarts" if we set $\beta = \max\{\beta^{\mathrm{PR}}, 0\}$. The event of β becoming negative generally happens when we have exhausted all the search directions, but we are not sure why.

Interestingly, if $A = I$ then we will never build up a basis of conjugate directions; but we won't need to, as the gradient will always point at the optimum.

4 Thoughts on extending NLCG

It is hard to think of a good multidimensional generalization of NLCG, as we are already effectively doing a Newton step at each NLCG reset. My March 18 2025 RT preprint speculates on such a generalization and proposes an algorithm based on a linear embedding, but this introduces additional overhead that seems unnecessary. NLCG is able to propose direction vectors one at a time, while still effectively inverting the Hessian, with a sequence of 1-dimensional Newton steps.

There must be a way to make it work "stochastically", i.e. with batches but presumably these should stay the same until the next reset. It could be trusted to make (close to) the smallest possible adjustment to the parameters to give the model good performance on the current batch. Then the NLCG resets would be like a heartbeat, and the current batch of data would be all the blood that fits into the "heart", and each reset would trigger a new batch to be called up. However this would be a problem with regularized models as the successive batches would fight with each other over the parameter vector's expressiveness.

A simple solution would be to create a kind of memory using $|\theta - \theta_0|$ regularization where θ_0 is from the end of the previous batch. The regularizer can be adjusted using the previous batch as a test set for example, and making slight adjustments up or down according to the $\frac{\mathrm{d}}{\mathrm{d}s}$ of the test error. Or we can curate a test set as we go using the RT. But we should have a traditional regularizer as well, and control the ratio of the regularization coefficients. And note that this can be done in a hierarchy; for example a separate parameter vector for the current hour, minute, and second.

[4]If we find ourselves in a concavity, then we might as well step out of it in a totally random direction. Although it is tempting to make use of the curvature, and for example take a step which is the exact opposite of the step that would take us to the local maximum.

NLCG should refuse to increase the objective. It is not clear what to do when this happens. If the gradient indicates the objective should decrease, then we can reduce the step size until we find a decrease from the current value. If however the Newton step on α is pointing us uphill, then this indicates we're in a local concavity, and it would make sense to try walking the same distance in the other direction before we even update x and calculate the new objective. In either case we may want to try larger and larger steps if the last step is successful, and smaller ones (or break out of the loop) if it isn't. Once a minimum is bracketed, the secant method or Newton's method can be used to pinpoint it, but it is not clear that this extra effort would be useful, as a reset would seem to be called for at the point where we find the objective increasing.

In the context of non-linear optimization, the (linear) conjugate gradients method can best be thought of as an algorithm for inverting the dual of the objective gradient with respect to the dual of the optimal independent variable (x^*). An input consists of an optimum x^*, an objective function, and a target dual gradient $-b$.

The algorithm finds an $\dot{x}$ such that $A\dot{x} = b$, which is to say that $\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\frac{\partial}{\partial x}f(x^* + \varepsilon\dot{x}) = \frac{\mathrm{d}}{\mathrm{d}\varepsilon}(A(x^* + \varepsilon\dot{x}) - b) = A\dot{x}$, which is the Hessian-vector product at x^* of f with $\dot{x}$, should equal the target value b. (In this setting, b is an input to the algorithm rather than part of the objective f) Each iteration updates $\dot{x}$ and computes a new HVP, always at the same $x = x^*$. It is also necessary to compute a HVP with the direction vector d in order to determine the step length α and the quotient coefficient β for the conjugate projection, although the latter can be approximated as a difference of successive gradients as in equation 17 to get the Fletcher-Reeves formula 30 with no HVP.

The CG algorithm can be used to apply the implicit function theorem to an optimization problem. Once an optimum x^* is found via NLCG, we can calculate total derivatives by using plain CG to approximate the inverse-HVP of Cauchy's implicit function theorem:

$$\frac{\mathrm{d}}{\mathrm{d}t}x^*(t) = -\left(\overbrace{\frac{\partial^2}{\partial x^2}f(x^*, t)}^{A}\right)^{-1} \overbrace{\frac{\partial^2}{\partial x \partial t}f(x^*, t)}^{b} \tag{35}$$

In this case b, which along with f is the input to the CG algorithm, is calculated as the derivative of the gradient $\frac{\partial f}{\partial x}$ with respect to the hyperparameter t. At first glance, if there is more than one hyperparameter or if t has more than one dimension, then it seems that multiple inverse-HVPs will be required, implying multiple runs of CG. However, when backpropagating adjoints through an optimization run, we can make use of the symmetry of A to make do with only a single inverse-HVP, involving the adjoint of x^*:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -\frac{\mathrm{d}y}{\mathrm{d}x^*}^\top \frac{\mathrm{d}}{\mathrm{d}t}x^* = \frac{\mathrm{d}y}{\mathrm{d}x^*}^\top \left(\frac{\partial^2 f}{\partial x^2}\right)^{-1} \frac{\partial^2 f}{\partial x \partial t} \tag{36}$$

$$= -\frac{\mathrm{d}}{\mathrm{d}t}\left(\underbrace{\left(A^{-1}\frac{\mathrm{d}y}{\mathrm{d}x^*}\right)^\top}_{\text{treat as constant}} \frac{\partial f}{\partial x}\right) \tag{37}$$

So the derivatives of all hyperparameters can be calculated by backpropagating adjoints from the "pseudo-objective" above, which is constructed from the function gradient and the inverse HVP of the incoming adjoint $\frac{\mathrm{d}y}{\mathrm{d}x^*}$. First use CG to calculate the inverse HVP $A^{-1}\frac{\mathrm{d}y}{\mathrm{d}x^*}$; then, treating this as a constant, take its inner product with the function gradient. Backpropagate adjoints from the resulting objective to all the hyperparameters t. We are not sure, but this must be related to the methods proposed in Lorraine, Vicol, Duvenaud, 2020, "Optimizing Millions of Hyperparameters by Implicit Differentiation". This is the key to an efficient "`back_opt_nlcg`" corresponding to backpropagation through NLCG optimization. For the corresponding forward-derivative "`dual_opt_nlcg`" function we can use the non-zero dual components to calculate $-b = \frac{\partial^2 f}{\partial x \partial t}$ (where t is the abstract dual variable) and pass this through A^{-1} in a single run of CG.