(Quasi-)Newton methods

Alexandre Gramfort

alexandre.gramfort@inria.fr

Master 2 Data Science, Univ. Paris Saclay
Optimisation for Data Science

Table of Contents

Outline

So far you have seen:

- gradient descent
- proximal gradient descent
- accelerated gradient descent
- (proximal) coordinate descent
- conjugate gradient

Now

- Newton methods
- Quasi-Newton methods
- Methods dedicated to non-linear least squares

Quasi-Newton and in particular L-BFGS are still heavily used to tackle smooth potentially large scale optim problems in machine learning (e.g. ℓ_2 logistic regression, conditional random fields)

Newton method

It is used to find the zeros of a differentiable non-linear function g:

$$\text{Find } x \text{ such that } g(x) = 0, \text{ where } g : \mathbb{R}^n \to \mathbb{R}^n.$$

Given a starting point x_0, Newton method consists in iterating:
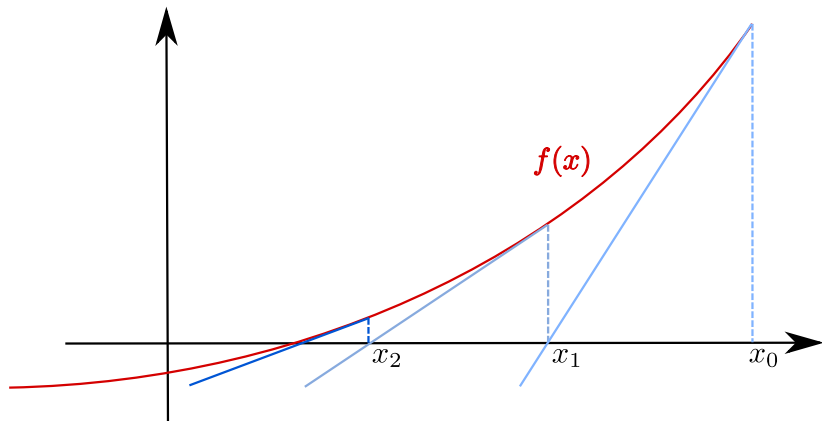
$$x_{k+1} = x_k - g'(x_k)^{-1} g(x_k)$$

where $g'(x)$ is the derivative (Jacobian) of g at point x.

We have that:

- $g'(x_k)$ is matrix in $\mathbb{R}^{n \times n}$
- each iteration requires to solve a linear system.

Newton method in 1d

Newton method?

Applying this method to the optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x)$$

consists in setting $g(x) = \nabla f(x)$, i.e., looking for stationary points. The iterations read:

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k) \ .$$

Newton method is particularly interesting as its convergence is quadratic locally around x^*, i.e.:

$$\|x_{k+1} - x^*\| \leq \gamma \|x_k - x^*\|^2, \gamma > 0 \ .$$

Convergence of Newton method

Theorem (Convergence of Newton method)

Let $g : \mathbb{R}^n \to \mathbb{R}^n$ assumed twice differentiable $\mathcal{C}^2$, and $x^* \in \mathbb{R}^n$ an isolated zero of g ($g(x^*) = 0$). Assuming that $g'(x^*)$ is invertible, there exists a closed ball $\mathcal{B}$ centered on x^*, such that for every $x_0 \in \mathcal{B}$, the sequence x_k obtained with Newton algorithm stays in $\mathcal{B}$ and converges towards x^*. Furthermore, there is a constant $\gamma > 0$, such that $\|x_{k+1} - x^*\| \leq \gamma \|x_k - x^*\|^2$.

Remark: Convergence of Newton is local. The method may diverge if the initial point is too far from x^* or if the Hessian is not positive definite. That is why Newton should be coupled with a line search strategy.

$\rightarrow$ See proof in lecture notes.

Newton on quadratic function

Exercise

Show that for a quadratic function

$$f(x) = \frac{1}{2}x^\top A x - b^\top x + c,\, x \in \mathbb{R}^n$$

with A symmetric positive definite, Newton method converges in one iteration independently of the choice of x_0.

Remark: Newton is therefore not affected by the conditioning of the problem (not like Gradient descent).

Table of Contents

Variable metric

The idea behind variable metric methods consists in using iterations of the form

$$
\begin{cases}
d_k = -B_k g_k \ , \\
x_{k+1} = x_k + \rho_k d_k \ ,
\end{cases}
$$

where $g_k = \nabla f(x_k)$, B_k is a positive definite matrix and $\rho_k \geq 0$ is a step size.

$\rightarrow$ If $B_k = I_n$, it corresponds to gradient descent.
$\rightarrow$ Setting $B_k = B$ is the fixed metric case.

Fixed metric case

When minimizing

$$\min_{x \in \mathbb{R}^n} f(x)$$

one can set $x = Cy$ with C invertible (change of variable).
Let us denote $\tilde{f}(y) = f(Cy)$. This leads to:

$$\nabla \tilde{f}(y) = C^\top \nabla f(Cy) \ .$$

Gradient descent applied to $\tilde{f}(y)$ reads:

$$y_{k+1} = y_k - \rho_k C^\top \nabla f(Cy_k)$$

which means using $B = CC^\top$ as it is equivalent to:

$$x_{k+1} = x_k - \rho_k CC^\top \nabla f(x_k) \ .$$

Question: How would you choose C for quadratic problem?

Quadratic case

Theorem (Preconditioned gradient descent)

Let $f(x)$ a positive definite quadratic form with Hessian A, and B a positive definite matrix. The preconditioned gradient algorithm:

$$\begin{cases} x_0 = \textit{fixed}, \\ x_{k+1} = x_k - \rho_k B g_k, \ \ \rho_k \ \textit{optimal} \end{cases}$$

has a linear convergence: $\|x_{k+1} - x^*\| \leq \gamma \|x_k - x^*\|$
where:

$$\gamma = \frac{\chi(BA) - 1}{\chi(BA) + 1} < 1 \ .$$

$\chi(M) = \lambda_1 / \lambda_n$ is the Euclidian conditioning *i.e.,* ratio of largest and lowest eigenvalues (≥ 1).

Quadratic case

So we have a linear convergence:

$$\|x_{k+1} - x^*\| \le \gamma \|x_k - x^*\|$$

where:

$$\gamma = \frac{\chi(BA) - 1}{\chi(BA) + 1} < 1 \ .$$

Remark

The lower the conditioning of BA, the faster is the algorithm. One cannot set $B = A^{-1}$ as it would imply having already solved the problem, but this however suggests to use B so that is approximate A^{-1}. This is the idea behind quasi-Newton methods.

Table of Contents

Quasi-Newton

A quasi-Newton method reads

$$
\begin{cases}
d_k = -B_k g_k \ , \\
x_{k+1} = x_k + \rho_k d_k \ ,
\end{cases}
$$

or

$$
\begin{cases}
d_k = -H_k^{-1} g_k \ , \\
x_{k+1} = x_k + \rho_k d_k \ ,
\end{cases}
$$

where B_k (resp. H_k) is a matrix which aims to approximate the inverse of the Hessian (resp. the Hessian) of f at x_k.

Question: How to achieve this?

Quasi-Newton

One can start with $B_0 = I_n$. how to update B_k at every iteration?

Idea: apply a Taylor expansion on the gradient, notice that at point x_k, the gradient and the Hessian are such that:

$$g_{k+1} = g_k + \nabla^2 f(x_k)(x_{k+1} - x_k) + \epsilon(x_{k+1} - x_k) \ .$$

If one assumes that the approximation is good enough one has:

$$g_{k+1} - g_k \approx \nabla^2 f(x_k)(x_{k+1} - x_k) \ ,$$

which leads to the quasi-Newton relation, a.k.a, the secant condition.

Quasi-Newton relation (or secant condition)

Definition (Quasi-Newton relation)

Two matrices B_{k+1} and H_{k+1} verify the quasi-Newton relation (or secant condition) if:

$$H_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$$

or

$$x_{k+1} - x_k = B_{k+1}(\nabla f(x_{k+1}) - \nabla f(x_k))$$

Problem: How to update B_k keeping it positive definite?

Update formula of Hessian

The update strategy at iteration

$$\begin{cases} d_k = -B_k g_k \ , \\ x_{k+1} = x_k + \rho_k d_k \ , \end{cases}$$

is to correct B_k with a symmetric matrix Δ_k:

$$B_{k+1} = B_k + \Delta_k$$

such that quasi-Newton relation holds:

$$x_{k+1} - x_k = B_{k+1}(g_{k+1} - g_k)$$

with B_{k+1} positive definite, assuming B_k is positive definite.
Idea: Use rank 1 or 2 matrices for Δ_k

Broyden formula (known as SR1)

Broyden formula is a rank 1 correction: $B_{k+1} = B_k + vv^\top$.
The matrix B_{k+1}, with $v \in \mathbb{R}^n$, should verify the quasi-Newton relation:

$$B_{k+1}y_k = s_k \,,$$

where $y_k = g_{k+1} - g_k$ and $s_k = x_{k+1} - x_k$. It follows that:

$$B_k y_k + vv^\top y_k = s_k$$
$$\Rightarrow (y_k^\top v)^2 = (s_k - B_k y_k)^\top y_k$$

Using the equality $vv^\top = \frac{vv^\top y_k (vv^\top y_k)^\top}{(v^\top y_k)^2}$,
one can write after replacing $vv^\top y_k$ by $s_k - B_k y_k$, and $(v^\top y_k)^2$ by $y_k^\top (s_k - B_k y_k)$, the correction formula

$$B_{k+1} = B_k + \frac{(s_k - B_k y_k)(s_k - B_k y_k)^\top}{(s_k - B_k y_k)^\top y_k} \,,$$

also known as Broyden formula.

Broyden formula

Theorem

Let f a quadratic form positive definite. Let us consider the method that, starting for x_0, iterates:

$$x_{k+1} = x_k + s_k \ ,$$

where the vectors s_k are linearly independent. Then the sequence of matrices starting by B_0 and defined as:

$$B_{k+1} = B_k + \frac{(s_k - B_k y_k)(s_k - B_k y_k)^\top}{(s_k - B_k y_k)^\top y_k} \ ,$$

where $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$, converges in less than n iterations towards A^{-1}, the inverse of the Hessian of f.

$\rightarrow$ Cf. proof in lecture notes

Remark: No guarantee that the matrices B_k are positive definite, even if the function f is quadratic and $B_0 = I_n$.

Using a rank 2 correction, it reads:

$$B_{k+1} = B_k + \alpha u u^\top + \beta v v^\top \ .$$

Imposing the quasi-Newton relation (secant condition):

$$B_{k+1} y_k = s_k$$
$$\Rightarrow B_k y_k + \alpha (u^\top y_k) u + \beta (v^\top y_k) v = s_k$$
$$\Rightarrow \alpha (u^\top y_k) u + \beta (v^\top y_k) v = s_k - B_k y_k$$

This equation has not a unique solution. The choice for u and v by DFP is:

$$u = s_k \quad \text{and} \quad v = B_k y_k$$

Solving for α and β the equation:

$$\alpha(s_k^\top y_k)s_k + \beta(y_k^\top B_k y_k)B_k y_k = s_k - B_k y_k$$

we obtain

$$\alpha = \frac{1}{s_k^\top y_k} \quad \text{and} \quad \beta = -\frac{1}{y_k^\top B_k y_k}$$

Davidon, Fletcher and Powell formula

The DFP formula is a rank 2 correction. It reads:

$$B_{k+1} = B_k + \frac{s_k s_k^\top}{s_k^\top y_k} - \frac{B_k y_k y_k^\top B_k}{y_k^\top B_k y_k} \quad . \tag{1}$$

Theorem

Let us consider the update

$$\begin{cases} d_k = -B_k g_k \ , \\ x_{k+1} = x_k + \rho_k B_k g_k, \ \rho_k \ optimal \end{cases}$$

where B_0 is positive definite and provided as well as x_0. Then the matrices B_k defined as in (1) are positive definite for all $k > 0$.

$\rightarrow$ Cf. proof in lecture notes

Davidon-Fletcher-Powell algorithm

Require: $\varepsilon > 0$ (tolerance), K (maximum number of iterations)

1: $x_0 \in \mathbb{R}^n$, $B_0 > 0$ (for example I_n)
2: **for** $k = 0$ to K **do**
3:     **if** $\|g_k\| < \varepsilon$ **then**
4:         break
5:     **end if**
6:     $d_k = -B_k \nabla f(x_k)$
7:     $x_{k+1} = x_k + \rho_k d_k$ (Compute optimal step size ρ_k)
8:     $s_k = \rho_k d_k$
9:     $y_k = g_{k+1} - g_k$
10:     $B_{k+1} = B_k + \frac{s_k s_k^\top}{s_k^\top y_k} - \frac{B_k y_k y_k^\top B_k}{y_k^\top B_k y_k}$
11: **end for**
12: **return** x_{k+1}

Davidon-Fletcher-Powell algorithm

This algorithm has a remarkable property when the function f is quadratic.

Theorem

When f is a quadratic form, the algorithm of Davidon-Fletcher-Powell generates a sequence of directions $s_0, \ldots, s_k$ which verify:

$$
\begin{aligned}
s_i A^\top s_j = 0, & \quad 0 \le i < j \le k, \\
B_{k+1} A s_i = s_i, & \quad 0 \le i \le k.
\end{aligned}
\tag{2}
$$

Remark: This theorem says that in the quadratic case, the algorithm is like a conjugate gradient method, which therefore converges in at most n iterations.

Remark: This required to have an optimal step size.

One can also notice that for $k = n - 1$

$$B_n A s_i = s_i, i = 0, \ldots, n - 1,$$

and since all s_i are linearly independent it implies $B_n = A^{-1}$.

Remark: One can show that in the general case (non-quadratic), if the direction d_k is reinitialized to $-g_k$ periodically, this algorithm converges to a local minimum $\hat{x}$ of f and that:

$$\lim_{k \to \infty} B_k = \nabla^2 f(\hat{x})^{-1} .$$

This implies that close to the optimum, the method behaves like a Newton method. This justifies the use of $\rho_k = 1$ when using approximate line search.

Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

- The BFGS formula is derived from the formula of DFP by swapping the roles of s_k and y_k.

- The formula obtained allows to maintain an approximation H_k of the Hessian which satisfies the same properties: $H_{k+1} > 0$ if $H_k > 0$ and satisfying the quasi-Newton relation:

$$y_k = H_k s_k \ .$$

- The BFGS formula therefore reads:

$$H_{k+1} = H_k + \frac{y_k y_k^\top}{y_k^\top s_k} - \frac{H_k s_k s_k^\top H_k}{s_k^\top H_k s_k} \ .$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

Require: $\varepsilon > 0$ (tolerance), K (maximum number of iterations)

1: $x_0 \in \mathbb{R}^n$, $H_0 > 0$ (for example I_n)

2: **for** $k = 0$ to K **do**

3:      **if** $\|g_k\| < \varepsilon$ **then**

4:          break

5:      **end if**

6:      $d_k = -H_k^{-1} \nabla f(x_k)$

7:      $x_{k+1} = x_k + \rho_k d_k$ (optimal step size ρ_k with line-search)

8:      $s_k = \rho_k d_k$

9:      $y_k = g_{k+1} - g_k$

10:     $H_{k+1} = H_k + \dfrac{y_k y_k^\top}{y_k^\top s_k} - \dfrac{H_k s_k s_k^\top H_k}{s_k^\top H_k s_k}$

11: **end for**

12: **return** x_{k+1}

Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

- Direction d_k is obtained by solving a linear system.
- In practice update of H_k is done on Cholesky factorization of $H_k = C_k C_k^\top$
- So the complexity of BFGS is the same as DFP.
- Cholesky factorization allows to check that H_k stays numerically positive definite

The BFGS algorithm has the same property as the DFP method:

- in the quadratic case it produces conjugate directions
- it converges in less than n iterations and $H_n = A$
- Usually combined with Wolfe and Powell's or Goldstein's rule.

but:

- much less sensitive than DFP to the use of approximate step size (to combine with Wolfe and Powell's or Goldstein's rule).

Remark: BFGS is in scipy see scipy . optimize . fmin_bfgs.

Limited-memory BFGS (L-BFGS) algorithm

- L-BFGS is a variant of BFGS that limits memory usage. It was originally proposed by Liu and Nocedal in 1989:
- Does not store matrix of the size of the Hessian, $n \times n$ which can be prohibitive in applications such as computer vision or machine learning where n can be millions.
- L-BFGS stores only a few vectors that are used to approximate the matrix H_k^{-1}
- So the memory usage is linear in the dimension of the problem.

[Liu, D. C.; Nocedal, J. (1989). "On the Limited Memory Method for Large Scale Optimization". Mathematical Programming B. 45 (3): 503528.]

Limited-memory BFGS (L-BFGS) algorithm

- L-BFGS is an algorithm of the quasi-Newton family with $d_k = -B_k \nabla f(x_k)$.
- Difference is in the computation of the product between B_k and $\nabla f(x_k)$.
- Idea is to keep in memory the last low rank corrections, more specifically the last m values of $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$.

Limited-memory BFGS (L-BFGS) algorithm

Let $\mu_k = \frac{1}{y_k^\top s_k}$, the algorithm to obtain d_k reads:

Require: m (memory size)

1: $q = g_k$
2: **for** $i = k-1$ to $k-m$ **do**
3:     $\alpha_i = \mu_i s_i^\top q$
4:     $q = q - \alpha_i y_i$
5: **end for**
6: $z = B_k^0 q$
7: **for** $i = k-m$ to $k-1$ **do**
8:     $\beta = \mu_i y_i^\top z$
9:     $z = z + s_i(\alpha_i - \beta)$
10: **end for**
11: $d_k = -z$

where B_k^0 is positive definite matrix, e.g., a diagonal matrix, so that initially setting z is fast.

Limited-memory BFGS (L-BFGS) algorithm

- Like BFGS, L-BFGS does not need exact line search to converge.
- L-BFGS is for smooth unconstrained problem but can be extended to handle simple box constraints (a.k.a. bound constraints): $l_i \leq x_i \leq u_i$ where l_i and u_i are per-variable constant lower and upper bounds. This algorithm called L-BFGS-B is due to Byrd et al. (1995).
- L-BFGS-B in scipy as scipy.optimize.fmin_l_bfgs_b.

[Byrd, R. H.; Lu, P.; Nocedal, J.; Zhu, C. (1995). "A Limited Memory Algorithm for Bound Constrained Optimization". SIAM J. Sci. Comput. 16 (5): 11901208. doi:10.1137/0916069.]

$\rightarrow$ Can you solve a Lasso with L-BFGS-B?

Table of Contents

The function to minimize reads:

$$f(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2 \ .$$

Newton method can be applied to the minimization of f. The gradient and the Hessian matrix read in this particular case:

$$\nabla f(x) = \sum_{i=1}^{m} f_i(x) \nabla f_i(x) \ ,$$

and

$$\nabla^2 f(x) = \sum_{i=1}^{m} \nabla f_i(x) \nabla f_i(x)^\top + \sum_{i=1}^{m} f_i(x) \nabla^2 f_i(x) \ .$$

Gauss-Newton method

Close to the optimum, the $f_i(x)$ are small so the second term can be ignored. The Hessian reads:

$$H(x) \approx \sum_{i=1}^{m} \nabla f_i(x) \nabla f_i(x)^\top .$$

This matrix is always positive. Furthermore when m is much larger than n, this matrix is often positive definite.

The Gauss-Newton method uses this approximation of $H(x)$ in a Newton-like solver:

$$\begin{cases} x_0 = \text{fixed}, \\ H_k = \sum_{i=1}^{m} \nabla f_i(x_k) \nabla f_i(x_k)^\top, \\ x_{k+1} = x_k - H_k^{-1} \nabla f(x_k) . \end{cases}$$

Gauss-Newton method

To guarantee the convergence of the Gauss-Newton method, it can
be combined with a line search procedure:

$$\begin{cases} x_0 = \text{fixed}, \\ H_k = \sum_{i=1}^{m} \nabla f_i(x_k) \nabla f_i(x_k)^\top, \\ x_{k+1} = x_k - \rho_k H_k^{-1} \nabla f(x_k) \ . \end{cases}$$

Levenberg-Marquardt method

- Levenberg-Marquardt method is a variant of Gauss-Newton that enforces that the Hessian approximation H_k is positive definite.

- The idea is simply to replace H_k by $H_k + \lambda I_n$.

$$\begin{cases} x_0 = \text{fixed}, \\ H_k = \sum_{i=1}^{m} \nabla f_i(x_k) \nabla f_i(x_k)^\top, \\ d_k = -(H_k + \lambda I_n)^{-1} \nabla f(x_k) \\ x_{k+1} = x_k + \rho_k d_k \ . \end{cases}$$

- If λ is large, method is equivalent to a gradient method.

- The Levenberg-Marquardt method in scipy as scipy.optimize.leastsq.

References

- Wright and Nocedal, Numerical Optimization, 1999, Springer, Chapters 6 and 8.