Optimization for Machine Learning: Second-order algorithms

Clément Royer

M2 IASD - 2019/2020

These lecture notes are meant to summarize and complement the contents of the lecture given on December 10, 2019.

The latest version can be accessed at the address
https://www.lamsade.dauphine.fr/~croyer/ensdocs/LectureNotes-OML-SecOrd.pdf

Current version: December 18, 2019.

Typos, questions, etc, can be signaled to: clement.royer@dauphine.psl.eu.

Notations

- $\mathbb{R}$ denotes the set of real numbers. The set of vectors with $n$ real components will be denoted by $\mathbb{R}^n$. The set of $m$-by-$n$ matrices with real coefficients will be denoted by $\mathbb{R}^{m \times n}$. Any vector $x \in \mathbb{R}^n$ will be considered as a column vector (that is, $\mathbb{R}^n \equiv \mathbb{R}^{n \times 1}$), and its corresponding row vector will be denoted by $x^T \in \mathbb{R}^{1 \times n}$.
- The set of nonnegative (resp. positive) real numbers will be denoted by $\mathbb{R}^+$ (resp. $\mathbb{R}^{++}$).
- The $i$-th component of a vector $x \in \mathbb{R}^n$ will be denoted by $x_i$.
- We write $x \geq 0$ to indicate that all components of a given vector must be nonnegative. Similarly, we may use $x > 0$, $x \leq 0$ or $x < 0$.
- The set of continuously differentiable (resp. twice continuously differentiable) functions from $\mathbb{R}^n$ to $\mathbb{R}$ will be denoted by $C^1$ (resp. $C^2$).
- Given a quantity $A$, the notation $O(A)$ means a constant times $A$, where the constant does not depend on $A$. The notation $\tilde{O}(A)$ means $O(A \log^c(A))$ for some $c > 0$. 

1
1 Motivation

In nonlinear optimization, it is known that exploiting second-order information can enhance the performance of first-order algorithms. Indeed, first-order methods such as gradient descent can be quite sensitive to the conditioning of the optimization problem, because such methods are not scale invariant.

Consider the problem \( \min_{w \in \mathbb{R}^d} f(w) \), where \( f \) is a continuously differentiable function, and a scaled version of the problem, \( \min_{\tilde{w} \in \mathbb{R}^d} f(A \tilde{w}) \), where \( A \in \mathbb{R}^{d \times d} \) is a linear invertible transformation of the input (typical of neural networks). The \( k \)-th iteration of gradient descent for the first problem is

\[
\begin{align*}
    w_{k+1} &= w_k - \alpha_k \nabla f(w_k),
\end{align*}
\]

while for the second problem, it is

\[
\begin{align*}
    \tilde{w}_{k+1} &= \tilde{w}_k - \alpha_k A \nabla f(A \tilde{w}_k).
\end{align*}
\]

If we set \( w_k = A \tilde{w}_k \) in order to obtain equivalent solutions, the second iteration becomes

\[
\begin{align*}
    w_{k+1} &= w_k - \alpha_k A^2 \nabla f(w_k),
\end{align*}
\]

which can have quite a different behavior than the original iteration, depending on the properties of the matrix \( A \).

Second-order methods, on the other hand, can be designed so as to be insensitive to such changes: Newton’s method is for instance invariant to linear transformations. This is one of the reasons why the optimization community started to focus on second-order methods in the 80’s.

1.1 Drawbacks of second-order methods

The cost of second-order methods has always been raised as an issue with these schemes. Indeed, a basic second-order method would require the computation of the Hessian matrix at the current point, along with the manipulation of that matrix in certain ways (solving a linear system, computing eigenvalues, etc). In the context of machine learning, this can be prohibitive due to the cost of accessing the data necessary for these calculations. In Section 4 we will describe several approaches that attempt to use second-order information at a reasonable cost.

If the problem at hand is not sufficiently smooth, it may also seem impossible to use second-order information. Like with first-order methods, equivalents of the second-order derivatives can be defined in multiple contexts, that allow for generalizing many second-order algorithms to this setting. This is particularly important in variational analysis, but is, however, out of the scope of this lecture.

---

1 This lead in particular to the development of interior-point methods, see the associated lecture notes.
2 Newton’s method

We begin by presenting the most classical second-order algorithm for optimization, Newton’s method. In order to provide justification for Newton’s method in optimization, we first recall its motivation in the context of nonlinear equations.

2.1 Newton’s method for nonlinear equations

This method was originally designed to solve systems of nonlinear equations. Considering the equation

\[ F(s) = 0, \quad F : \mathbb{R}^n \rightarrow \mathbb{R}^n \]  (2.1)

and assuming that \( F \) is continuous, Newton’s method attempts to find a solution of (2.1) through the iteration

\[ s_{k+1} = s_k - J(s_k)^{-1}F(s_k), \]

where \( J(s) = \left[ \frac{\partial F}{\partial w_j}(w) \right]_{i=1,...,d} \). Note that this iteration assumes that the Jacobian can be inverted.

Note also that no stepsize is used in the iteration. Rather, the iteration is built so that the inversion of the Jacobian matrix provides a scaling of the components of \( F \) according to the derivative information.

2.2 Newton’s method in nonlinear optimization

In nonlinear optimization, we consider \( \min_{w \in \mathbb{R}^d} f(w) \) where \( f \in C^2 \). In that context, Newton’s method corresponds to applying Newton’s method for nonlinear equations to the system \( \nabla f(w_k) = 0 \). The iteration of Newton’s method thus is

\[ w_{k+1} = w_k - \left[ \nabla^2 f(w_k) \right]^{-1} \nabla f(w_k), \]  (2.2)

whenever the Hessian matrix of \( f \) at \( w_k \) is not singular. This is for instance the case when \( f \) is a strongly convex function.

Two equivalent formulations of Newton’s method are of importance. The first one does not explicitly use the inverse of the Hessian matrix (which may not exist if the function is not strongly convex):

\[ w_{k+1} = w_k + s_k, \quad \nabla^2 f(w_k)s_k = -\nabla f(w_k). \]  (2.3)

Note that the linear system \( \nabla^2 f(w_k)s_k = -\nabla f(w_k) \) is commonly called the Newton system or the Newton equations. Solutions of this system can exist even when the Hessian matrix is indefinite.

The second one formulates the Newton step as a solution of a quadratic problem:

\[ w_{k+1} = \arg \min_{s \in \mathbb{R}^d} f(w_k) + \nabla f(w_k)^T s + \frac{1}{2}s^T \nabla^2 f(w_k)s. \]  (2.4)

This problem may not have a solution when \( f \) is not convex.

2.3 Local convergence of Newton’s method

One of the main characteristics of Newton’s method is its fast local convergence rate guarantees. When \( f \) is a strongly convex quadratic, Newton’s method converges in one iteration. If \( f \) is
strongly convex, it can be shown that Newton’s method converges at a local quadratic rate, i.e. if \( w_0 \) is close enough to the global minimum \( w^* \) of \( f \), we have:

\[
\|w_{k+1} - w^*\| \leq C \|w_k - w^*\|^2,
\]

where \( C > 0 \). This reflects on the practical performance of Newton’s method, which commonly takes only a few iterations to converge on such functions.

Note that similar rates can also be obtained for convex or nonconvex functions, provided the function is strongly convex in a neighborhood of a local minimum: this illustrates that this result is essentially local (and shows the importance of a good initialization).

3 Globalization techniques

As described in the previous section, Newton’s method possesses local convergence guarantees. It can be shown that those guarantees are not global, in the sense that for general nonconvex functions (or even convex functions that are not strongly convex), the choice of the initial point determines whether or not the method will converge, or be well-defined. In practice, finite-precision arithmetic can even prevent Newton from converging on strongly convex functions.

Globalization techniques were developed to guarantee that Newton’s method would converge independently of its starting point, and regardless of the convexity of the problem. They have proven to be quite useful in multiple settings, even though they rely on seemingly costly operations at every iteration (compute minimum eigenvalue, solve a minimization problem, etc). We present thereafter the three main approaches for globally convergent Newton methods, then discuss their theoretical properties.

3.1 Line search

Line-search algorithms proceed by first computing a suitable direction of decrease for the objective function, then by selecting an appropriate stepsize along this direction. Line-search Newton methods construct this direction by modifying the Newton system \([2, 3]\) so that it has a solution.

A natural idea to guarantee that a step can be uniquely obtained from the Newton equations consists in regularizing the Hessian so that it becomes positive definite. At every iteration \( k \), one can indeed select a nonnegative value \( \lambda_k \) such that \( \nabla^2 f(w_k) + \lambda_k I > 0 \) and compute a (regularized) Newton step as \( s_k = -\left[\nabla^2 f(w_k) + \lambda_k I\right]^{-1}\nabla f(w_k) \).

Once the direction has been set, a line-search process is executed to compute the most appropriate steplength along that direction. In the case of Newton-type methods, it is usually a good idea to try the unit step first (especially if \( \lambda_k = 0 \)). A classical process, called backtracking, tries geometrically decreasing values (e.g. \( 1, 1/2, 1/4, \ldots \)) until finding a value that sufficiently decreases the objective. Algorithmically, this means that given \( s_k \), the method determines \( \alpha_k > 0 \) such that (at least) \( f(w_k + \alpha_k s_k) < f(w_k) \), then set \( w_{k+1} = w_k + \alpha_k s_k \).

With more precise definitions, it is possible to show convergence of this framework regardless of the starting point. Under some additional conditions, it is also possible to obtain local convergence results, that are usually worse than Newton’s method because of the use of \( \lambda_k \). Still, line-search methods are widely used in nonlinear optimization, and Newton-type directions turn out to be less sensitive to the choice of stepsize than gradient-based methods.
3.2 Trust region

In the optimization community, trust-region methods are often preferred to line-search methods for handling nonconvex problems. Rather than regularizing the Newton system, those techniques modify the subproblem formulation of Newton’s method (2.4). At every iteration, the step $s_k$ is thus obtained as the solution of a trust-region subproblem:

$$s_k = \arg \min_{s \in \mathbb{R}^d} \nabla f(w_k)^T s + \frac{1}{2} s^T \nabla^2 f(w_k) s \quad \text{subject to} \quad \|s\| \leq \delta_k. \quad (3.1)$$

The parameter $\delta_k$ is called the trust-region radius, and prevents the solution to go to infinity if the Hessian of $f$ at $w_k$ is indefinite. Depending on whether or not $s_k$ leads to a decrease in the function value, the step is accepted or rejected, and $\delta_k$ is increased or decreased.

Trust-region algorithms have been endowed with similar global convergence guarantees than line-search methods and can even have faster local convergence rates, if the management of the trust-region radius allows for taking Newton steps close to a minimum.

3.3 Cubic regularization

This more recent technique has gained popularity because of its attractive complexity properties. It proceeds similarly to the trust-region paradigm, by choosing $s_k$ as the solution to the subproblem

$$s_k = \arg \min_{s \in \mathbb{R}^d} \nabla f(w_k)^T s + \frac{1}{2} s^T \nabla^2 f(w_k) s + \frac{\sigma_k}{3} \|s\|^3. \quad (3.2)$$

The parameter $\sigma_k > 0$ ensures that this subproblem always has a finite solution: this value of $\sigma_k$ is updated in an inverse way compared to the trust-region radius. Note that cubic regularization can be seen as an implicit regularization of the Hessian, that is not known before $s_k$ is computed. On the contrary, the regularizing parameter in the line-search framework is known before computing the step.

3.4 Convergence and complexity

Under appropriate assumptions, these three methods can be shown to converge to a stationary point in the nonconvex and convex cases. In a convex setting (and even more so in a strongly convex setting), local convergence results are usually of interest, and Newton-type methods can be designed to take advantage of Newton steps for fast local rates.

In the nonconvex case, recent research has focused on bounding the number of iterations required to satisfy $\|\nabla f(w_k)\| \leq \epsilon$ for $\epsilon \in (0, 1)$. In their standard versions, the line-search and trust-region variants require at most $O(\epsilon^{-2})$ iterations to reach such a point, while cubic regularization requires at most $O(\epsilon^{-3/2})$ iterations (this is the optimal bound for Newton-type methods). This has motivated a significant amount of research on cubic regularization techniques, even though those turned out to be less efficient in practice than trust-region methods (in part because solving cubic subproblems can be harder than solving trust-region subproblems).
4 Practical second-order methods

In their most basic form, Newton’s method and its variants assume that the exact Hessian matrix is available, which is not reasonable in a number of settings (including machine learning, but also many physical-based applications like control). For this reason, inexact variants of Newton’s method have been developed, that do not require full storage or computation of the Hessian.

4.1 Hessian-free inexact Newton methods

One approach that has become quite popular in large-scale optimization is to compute an inexact Newton step by approximately solving the Newton equations at every iteration. That is, at iteration $k$, the step $s_k$ is obtained such that

$$\|\nabla^2 f(w_k)s_k + \nabla f(w_k)\| \leq \eta_k,$$  \hspace{1cm} (4.1)

where $\eta_k > 0$ (with $\eta_k = 0$, we would recover the exact Newton iteration). One advantage of using (4.1) is that it can be satisfied even when the linear system does not have a solution. Moreover, it allows for applying iterative linear algebra techniques to the linear system, that are matrix-free in nature: they do not require access to the full matrix, only to products of this matrix with a vector. One example of such an algorithm is the (linear) conjugate gradient method, that is guaranteed to solve a positive definite linear system of dimension $d$ in $d$ iterations in exact arithmetic. In practice, such a method is applied for a certain number of iterations, and stops when (4.1) is satisfied or this budget is exhausted.

Computing Hessian-vector products. With Hessian-free techniques, all that is required is the computation of $\nabla^2 f(w_k)v$ for any $v \in \mathbb{R}^d$. One possibility to obtain these quantities without access to the Hessian is to use finite-difference estimators based on several gradient evaluations. If this is still too expensive, automatic differentiation techniques might be useful. If a code for computing the numerical value of a gradient is available, automatic differentiation obtains numerical values of Hessian-vector products at a cost that is only more expensive than a gradient evaluation by a constant factor. Automatic differentiation is used in machine learning packages such as PyTorch for computing gradients, and can also be employed to calculate Hessian-vector products.

4.2 Subsampling Hessian-free methods

Consider the following finite-sum optimization problem:

$$\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w).$$  \hspace{1cm} (4.2)

In data-driven applications, it is quite common to encounter problems of this form, for which each $f_i$ depends on one data point. As a result, evaluating the entire function $f$ (or its derivatives) is extremely costly, and practitioners rely on subsampling techniques, that draw a generally small subset of the data points on which derivatives are calculated.

\footnote{A similar analysis holds for general stochastic optimization problems of the form $f(w) = E_\xi [f(w; \xi)]$.}
This idea can be applied to inexact Newton methods as well. At every iteration $k$ of the algorithm, given the current iterate $w_k$, one computes subsampling derivatives as

$$
\nabla f_{S_k}(w_k) = \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(w_k), \quad \nabla^2 f_{S_k^H}(w_k) = \frac{1}{|S_k^H|} \sum_{i \in S_k^H} \nabla^2 f_i(w_k),
$$

(4.3)

where $S_k$ and $S_k^H$ are drawn randomly in $\{1, \ldots, n\}$. One can then apply an inexact Newton step approach to the system defined by these derivatives. For instance, a Newton-conjugate gradient method would apply the linear conjugate gradient algorithm to the linear system

$$
\nabla^2 f_{S_k^H}(w_k)s = -\nabla f_{S_k}(w_k)
$$

until a budget of conjugate gradient iterations has been exhausted, or a vector $s$ is found such that

$$
\|\nabla^2 f_{S_k^H}(w_k)s_k + \nabla f_{S_k}(w_k)\| \leq \rho\|\nabla f_{S_k}(w_k)\|
$$

(4.4)

is satisfied. As for the classical Newton-CG approach, globalization techniques would then be used to guarantee convergence even for nonconvex functions. In order for such guarantees to hold, the sample sizes $|S_k|$ and $|S_k^H|$ must be sufficiently large, which contrasts with the typical choices adopted in practice. In particular, if $|S_k|$ is not large enough, it is likely that the noise within the gradient estimation will be amplified by the Newton step, and that the resulting step will not be informative; still, small sample sizes can lead to good practical performance.

**Computing subsampled Hessian-vector products** In addition to the aforementioned techniques, the particular form of the objective function can be used to compute Hessian-vector products efficiently. Considering for instance a logistic loss objective of the form

$$
f(w) = \frac{1}{n} \sum_{i=1}^{n} \log \left(1 + \exp(-y_i w^T x_i)\right),
$$

we obtain the following formula for subsampled Hessian-vector products:

$$
\nabla^2 f_{S_k^H}(w_k)d = \frac{1}{|S_k^H|} \sum_{i \in S_k^H} \frac{\exp(-y_i w^T x_i)}{(1 + \exp(-y_i w^T x_i))^2}(x_i^T d)x_i.
$$

The field of subsampling Newton methods is an active area of research, particularly in the optimization community, but reconciling theory and practice remains an open question.

### 4.3 Quasi-Newton methods

Along with inexact Newton techniques, *quasi-Newton schemes* have been quite successful in large-scale optimization problems (and even when second-order derivatives do not exist!). Because of the way they approximate second-order information without computing Hessian values, they have also been favored from a practical viewpoint.

For the problem $\min_{w \in \mathbb{R}^d} f(w)$, the $k$-th iteration of a quasi-Newton method typically has the following form:

$$
w_{k+1} = w_k - \alpha_k H_k \nabla f(w_k),
$$

(4.5)

where $H_k$ is a symmetric, positive-definite matrix such that $H_k^{-1} \approx \nabla^2 f(w_k)$. The quasi-Newton matrix $H_k$ is updated dynamically at every iteration. Several formulas for such an
update have been proposed in the literature, such as the DFP (Davidon-Fletcher-Powell) and BFGS (Broyden-Fletcher-Goldfarb-Shanno), the latter of which is detailed below.

**BFGS Quasi-Newton** Start with $\mathbf{w}_0$ and $\mathbf{H}_0 = \mathbf{I}$ (the identity matrix). At every iteration $k$, compute $\mathbf{w}_{k+1}$ according to (4.5), then define $\mathbf{s}_k = \mathbf{w}_{k+1} - \mathbf{w}_k$ and $\mathbf{v}_k = \nabla f(\mathbf{w}_{k+1}) - \nabla f(\mathbf{w}_k)$. Finally, update the quasi-Newton matrix as follows:

$$
\mathbf{H}_{k+1} = \left( \mathbf{I} - \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{v}_k} \right)^T \mathbf{H}_k \left( \mathbf{I} - \frac{\mathbf{v}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{v}_k} \right) + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{v}_k}.
$$

(4.6)

Such an update guarantees that $\mathbf{H}_{k+1}^{-1} \mathbf{s}_k = \mathbf{v}_k$: that is, the new Hessian formula is consistent with the most recent displacement (such a relation is often called a *secant equation*).

In the case of a strongly convex function $f$, it can be shown that quasi-Newton methods such as BFGS possess a *local superlinear rate*. That is, for $\mathbf{w}_0$ sufficiently close to the optimum $\mathbf{w}^*$, the iterates satisfy the relation

$$
\|\mathbf{w}_{k+1} - \mathbf{w}^*\| \leq C \|\mathbf{w}_k - \mathbf{w}^*\|^{1+t},
$$

where $t \in (0, 1)$, $C > 0$ and $\mathbf{w}^*$ is the global minimum of the function.

**L-BFGS** A quasi-Newton matrix is likely to become dense as the iteration unfolds, and this can be an issue in a large-dimensional setting. The *limited-memory* variant of the BFGS Quasi-Newton update removes the need for storing a matrix $\mathbf{H}_k$, by considering the BFGS updates (4.6) in a recursive fashion. Indeed, the formula for $\mathbf{H}_{k+1}$ only depends on $\mathbf{H}_0$ and the pairs $(\mathbf{s}_0, \mathbf{v}_0), (\mathbf{s}_1, \mathbf{v}_1), \ldots, (\mathbf{s}_k, \mathbf{v}_k)$. Instead of performing the $k+1$ updates, the limited-memory L-BFGS update only computes $\mathbf{H}_{k+1}$ based on the latest $m$ pairs $\{(\mathbf{s}_{k-i}, \mathbf{v}_{k-i})\}_{i=0}^{\text{max}(0, m-1)}$. This can be implemented very efficiently: the value $m = 5$ is commonly used in implementations, and is sufficient to observe significant improvement compared to a gradient descent method.

A remarkable property of quasi-Newton methods is that they were used for quite some time before a convergence proof became available, because they performed remarkably well in practice. This performance continues to be demonstrated with variants such as L-BFGS, that can be extremely efficient on data science problems arising from robust statistics and matrix approximation. It would seem that a full explanation for this performance, in particular regarding the limited memory variants, remains to be found.

For problems exhibiting a structure similar to (4.2), it is possible to implement *subsampled quasi-Newton methods*, that only depend on subsampled gradients. For this reason, these methods require smaller batch sizes that what is theoretically needed for Hessian-free Newton methods (typically subsampling L-BFGS with $m = 5$ would be more expensive than stochastic gradient by a factor of 20).

---

3William C. Davidon (1927–2013) is credited for the original idea behind quasi-Newton methods in the late 50s. As a physicist, he applied this technique because he could not afford to compute second-order derivatives in his application. His work was celebrated in 1991, when the first issue of *SIAM Journal on Optimization* published his original technical report, that was rejected by a journal at the time Davidon submitted it, and that he had given up on publishing. Davidon also has an interesting backstory, that can be found on his Wikipedia page.
4.4 Gauss-Newton methods

By using the structure of a given problem, it is generally possible to use more information. This is a general optimization principle that turns out to be quite important for machine learning (the most prominent example being exploiting finite-sum structures). Here we present a way to extract second-order information from gradient information when the optimization problem possesses a particular formulation.

Consider the problem \( \min_{w \in \mathbb{R}^d} f(w) \), where \( f \) is a least-squares objective of the form

\[
f(w) = \frac{1}{2} \| \phi(w) \|^2,
\]

with \( \phi = [\phi_j]_{j=1}^m \in C^2(\mathbb{R}^d, \mathbb{R}^m) \). The gradient and Hessian of \( f \) are given by the following formula:

\[
\nabla f(w) = J_{\phi}(w)^T \phi(w), \quad \nabla^2 f(w) = J_{\phi}(w)^T J_{\phi}(w) + \sum_{j=1}^m \phi_j(w) \nabla^2 \phi_j(w),
\]

(4.7)

where \( J_{\phi}(w) = \begin{bmatrix} \nabla \phi_1(w)^T \\ \vdots \\ \nabla \phi_m(w)^T \end{bmatrix} \) is the Jacobian matrix of \( \phi \) at \( w \). Because this Jacobian is needed for computing \( \nabla f \), the first term of \( \nabla^2 f \) is available “for free” once the gradient has been computed. In addition, if the solution of the problem \( w^* \) is such that \( \phi(w^*) = 0 \), the second term in the formula of \( \nabla^2 f \) can be neglected around the solution. This is the underlying idea of Gauss-Newton methods, that rely on the following iteration:

\[
w_{k+1} = w_k + s_k, \quad J_{\phi}(w_k)^T J_{\phi}(w_k) s_k = J_{\phi}(w_k) \phi(w_k).
\]

(4.8)

or a globalized variant thereof. The matrix \( J_{\phi}(w_k)^T J_{\phi}(w_k) \) is called the Gauss-Newton approximation to the Hessian matrix. Although Gauss-Newton techniques lack the fast convergence rate guarantees of Newton variants near an optimum, they can lead to better steps away from the solution.

Gauss-Newton techniques have been applied to finite-sum problems with differentiable losses, of the following form:

\[
\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{n} \sum_{i=1}^n \ell(h(x_i; w), y_i)),
\]

(4.9)

where \( h : \mathbb{R}^d \rightarrow \mathbb{R}^d \) and \( \ell : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^d \) are both twice continously differentiable. In that case, one usually selects a batch \( S_k \) for the gradient and a batch \( S_k^H \) for the Gauss-Newton approximation of the Hessian: the latter is given by

\[
\frac{1}{|S_k^H|} \sum_{i \in S_k^H} J_h(x_i; w)^T \frac{\partial^2 \ell(h(x_i; w), y_i)}{\partial w^2} J_h(x_i; w).
\]

4.5 Diagonal scaling

We mention in passing another kind of second-order methods, that resembles the quasi-Newton idea. The iteration takes the same form as (4.5), but the matrix \( H_k \) is required to be diagonal.
This means that a different scaling is applied to every component of the gradient vector. In the optimization community, one of the most classical methods of this form is the Barzilai-Borwein method.

Closer to machine learning applications, some approaches based on using the diagonal of a (subsampled) Gauss-Newton-type matrix have also been proposed. It is worth noticing that this idea resembles the concept of batch normalization, that can be successfully inserted in layers of a neural network.

5 Conclusion

Second-order methods have been widely used in numerical analysis (PDEs, optimization, control) on very large-scale instances, but are not as common in data-driven applications. Nevertheless, there is an undeniable practical value in incorporating second-order aspects, not necessarily at the cost of evaluating second-order derivatives, and this trend of research is still being explored in a variety of data science problems.

References


