WIKIPEDIA

Limited-memory BFGS

Limited-memory BFGS (**L-BFGS** or **LM-BFGS**) is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm using a limited amount of computer memory. It is a popular algorithm for parameter estimation in machine learning.[1][2]

The algorithm's target problem is to minimise $f(\mathbf{x})$ over unconstrained values of the real-vector $\mathbf{x}$ where f is a differentiable scalar function.

Like the original BFGS, L-BFGS uses an estimation to the inverse Hessian matrix to steer its search through variable space, but where BFGS stores a dense $n \times n$ approximation to the inverse Hessian (n being the number of variables in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. Due to its resulting linear memory requirement, the L-BFGS method is particularly well suited for optimization problems with a large number of variables. Instead of the inverse Hessian $\mathbf{H}_k$, L-BFGS maintains a history of the past m updates of the position $\mathbf{x}$ and gradient $\nabla f(\mathbf{x})$, where generally the history size m can be small (often $m<10$). These updates are used to implicitly do operations requiring the $\mathbf{H}_k$-vector product.

Contents

Algorithm

The algorithm starts with an initial estimate of the optimal value, $\mathbf{x}_0$, and proceeds iteratively to refine that estimate with a sequence of better estimates $\mathbf{x}_1, \mathbf{x}_2, \ldots$. The derivatives of the function $g_k := \nabla f(\mathbf{x}_k)$ are used as a key driver of the algorithm to identify the direction of steepest descent, and also to form an estimate of the Hessian matrix (second derivative) of f.

L-BFGS shares many features with other quasi-Newton algorithms, but is very different in how the matrix-vector multiplication for finding the search direction is carried out $d_k = -H_k g_k$, where g_k is the current derivative and H_k is the inverse of the Hessian matrix. There are multiple published approaches using a history of updates to form this direction vector. Here, we give a common approach, the so-called "two loop recursion."[3][4]

We'll take as given x_k, the position at the k-th iteration, and $g_k \equiv \nabla f(x_k)$ where f is the function being minimized, and all vectors are column vectors. We also assume that we have stored the last m updates of the form $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$. We'll define $\rho_k = \frac{1}{y_k^{\mathrm{T}} s_k}$, and H_k^0 will be the 'initial' approximate of the inverse Hessian that our estimate at iteration k begins with.

The algorithm is based on the BFGS recursion for the inverse Hessian as

$$H_{k+1} = (I - \rho_k s_k y_k^\top) H_k (I - \rho_k y_k s_k^\top) + \rho_k s_k s_k^\top.$$

For a fixed k we define a sequence of vectors $q_{k-m}, \ldots, q_k$ as $q_k := g_k$ and $q_i := (I - \rho_i y_i s_i^\top) q_{i+1}$. Then a recursive algorithm for calculating q_i from q_{i+1} is to define $\alpha_i := \rho_i s_i^\top q_{i+1}$ and $q_i = q_{i+1} - \alpha_i y_i$.

We also define another sequence of vectors $z_{k-m}, \ldots, z_k$ as $z_i := H_i q_i$. There is another recursive algorithm for calculating these vectors which is to define $z_{k-m} = H_k^0 q_{k-m}$ and then recursively define $\beta_i := \rho_i y_i^\top z_i$ and $z_{i+1} = z_i + (\alpha_i - \beta_i) s_i$. The value of z_k is then our approximation for the direction of steepest ascent.

Thus we can compute the (uphill) direction as follows:

$q = g_k$
For $i = k-1, k-2, \ldots, k-m$
    $\alpha_i = \rho_i s_i^\top q$
    $q = q - \alpha_i y_i$
$H_k^0 = y_{k-1} s_{k-1}^\top / y_{k-1}^\top y_{k-1}$
$z = H_k^0 q$
For $i = k-m, k-m+1, \ldots, k-1$
    $\beta_i = \rho_i y_i^\top z$
    $z = z + s_i(\alpha_i - \beta_i)$
Stop with $H_k g_k = z$

This formulation is valid whether we are minimizing or maximizing. Note that if we are minimizing, the search direction would be the negative of z (since z is "uphill"), and if we are maximizing, H_k^0 should be negative definite rather than positive definite. For minimization, the inverse Hessian H_k^0 must be positive definite. The matrix is thus often represented as a diagonal matrix, with the added benefit that initially setting z only requires element-by-element multiplication.

The scaling of the initial matrix $\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$ ensures that the search direction is well scaled and therefore the unit step length is accepted in most iterations. A Wolfe line search is used to ensure that the curvature condition is satisfied and the BFGS updating is stable. Note that some software implementations use an Armijo backtracking line search, but cannot guarantee that the curvature condition $y_k^T s_k > 0$ will be satisfied by the chosen step since a step length greater than 1 may be needed to satisfy this condition. Some implementations address this by skipping the BFGS update when $y_k^T s_k$ is negative or too close to zero, but this approach is not generally recommended since the updates may be skipped too often to allow the Hessian approximation H_k to capture important curvature information.

This two loop update only works for the inverse Hessian. Approaches to implementing L-BFGS using the direct approximate Hessian B_k have also been developed, as have other means of approximating the inverse Hessian.[5]

Applications

L-BFGS has been called "the algorithm of choice" for fitting log-linear (MaxEnt) models and conditional random fields with ℓ_2-regularization.[2]

Variants

Since BFGS (and hence L-BFGS) is designed to minimize smooth functions without constraints, the L-BFGS algorithm must be modified to handle functions that include non-differentiable components or constraints. A popular class of modifications are called active-set methods, based on the concept of the active set. The idea is that when restricted to a small neighborhood of the current iterate, the function and constraints can be simplified.

L-BFGS-B

The **L-BFGS-B** algorithm extends L-BFGS to handle simple box constraints (aka bound constraints) on variables; that is, constraints of the form $l_i \leq x_i \leq u_i$ where l_i and u_i are per-variable constant lower and upper bounds, respectively (for each x_i, either or both bounds may be omitted).[6][7] The method works by identifying fixed and free variables at every step (using a simple gradient method), and then using the L-BFGS method on the free variables only to get higher accuracy, and then repeating the process.

OWL-QN

Orthant-wise limited-memory quasi-Newton (**OWL-QN**) is an L-BFGS variant for fitting ℓ_1-regularized models, exploiting the inherent sparsity of such models.[2] It minimizes functions of the form

$$f(\vec{x}) = g(\vec{x}) + C\|\vec{x}\|_1$$

where g is a differentiable convex loss function. The method is an active-set type method: at each iterate, it estimates the sign of each component of the variable, and restricts the subsequent step to have the same sign. Once the sign is fixed, the non-differentiable $\|\vec{x}\|_1$ term becomes a smooth linear term which can be handled by L-BFGS. After an L-BFGS step, the method allows some variables to change sign, and repeats the process.

O-LBFGS

Schraudolph *et al.* present an online approximation to both BFGS and L-BFGS.[8] Similar to stochastic gradient descent, this can be used to reduce the computational complexity by evaluating the error function and gradient on a randomly drawn subset of the overall dataset in each iteration. It has been shown that O-LBFGS has a global almost sure convergence [9] while the online approximation of BFGS (O-BFGS) is not necessarily convergent.[10]

Implementations

An early, open source implementation of L-BFGS in Fortran exists in Netlib as a shar archive [1] (http://netlib.org/opt/lbfgs_um.shar). Multiple other open source implementations have been produced as translations of this Fortran code (e.g. java (http://riso.sourceforge.net/), and python (http://www.scipy.org/doc/api_docs/SciPy.optimize.lbfgsb.html#fmin_l_bfgs_b) via SciPy). Other implementations exist:

- fmincon (Matlab optimization toolbox) (http://www.mathworks.com/help/toolbox/optim/ug/fmincon.html)
- FMINLBFGS (for Matlab, BSD license) (http://www.mathworks.com/matlabcentral/fileexchange/23245)
- minFunc (also for Matlab) (http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html)
- LBFGS-D (in the D programming language) (https://github.com/AdRoll/lbfgs-d))
- Frequently as part of generic optimization libraries (e.g. Mathematica (http://reference.wolfram.com/mathematica/tutorial/UnconstrainedOptimizationQuasiNewtonMethods.html), FuncLib C# library (http://funclib.codeplex.com/), and dlib C++ library (http://dlib.net/optimization.html))
- The libLBFGS (http://www.chokkan.org/software/liblbfgs/) is a C implementation.
- Maximization in Two-Class Logistic Regression (https://msdn.microsoft.com/library/azure/b0fd7660-eeed-43c5-9487-20d9cc79ed5d#bkmk_Notes) (in Microsoft Azure ML (https://azure.microsoft.com/en-us/services/machine-learning/))

Implementations of variants

The L-BFGS-B variant also exists as ACM TOMS (http://toms.acm.org/) algorithm 778.[7] In February 2011, some of the authors of the original L-BFGS-B code posted a major update (version 3.0).

A reference implementation[11] is available in Fortran 77 (and with a Fortran 90 interface) at the author's website (http://users.eecs.northwestern.edu/~nocedal/lbfgsb.html). This version, as well as older versions, has been converted to many other languages, including a Java wrapper (http://www.mini.pw.edu.pl/~mkobos/programs/lbfgsb_wrapper/index.html) for v3.0; Matlab interfaces for v3.0 (http://www.mathworks.com/matlabcentral/fileexchange/35104-lbfgsb-l-bfgs-b-mex-wrapper), v2.4 (http://www.cs.toronto.edu/~liam/software.shtml), and v2.1 (http://www.cs.ubc.ca/~pcarbo/lbfgsb-for-matlab.html); a C++ interface (http://code.google.com/p/otkpp/source/browse/trunk/otkpp/localsolvers/lbfgsb/LBFGSB.cpp?r=51) for v2.1; a Python interface for v3.0 as part of scipy.optimize.minimize (http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html); an OCaml interface (http://forge.ocamlcore.org/projects/lbfgs/) for v2.1 and v3.0; version 2.3 has been converted to C by f2c and is available at this website (http://www.koders.com/c/fid4A53890DFB42BB9734639793C7BDD4EB1B8E6583.aspx?s=decomposition); and R's `optim` general-purpose optimizer routine includes L-BFGS-B by using `method="L-BFGS-B"`.[12]

There exists a complete C++11 rewrite of the L-BFGS-B solver (https://github.com/PatWie/LBFGSB) using Eigen3.

OWL-QN implementations are available in:

- C++ implementation by its designers (http://research.microsoft.com/en-us/downloads/b1eb1016-1738-4bd5-83a9-370c9d498a03/), includes the original ICML paper on the algorithm[2]
- The CRF toolkit Wapiti (http://wapiti.limsi.fr) includes a C implementation
- libLBFGS

Works cited

1. Malouf, Robert (2002). *A comparison of algorithms for maximum entropy parameter estimation* (https://web.archive.org/web/20131101205929/http://acl.ldc.upenn.edu/W/W02/W02-2018.pdf) (PDF). Proc. Sixth Conf. on Natural Language Learning (CoNLL). pp. 49–55. Archived from the original (http://acl.ldc.upenn.edu/W/W02/W02-2018.pdf) (PDF) on 2013-11-01.
2. Andrew, Galen; Gao, Jianfeng (2007). "Scalable training of L_1-regularized log-linear models". *Proceedings of the 24th International Conference on Machine Learning* (http://research.microsoft.com/apps/pubs/default.aspx?id=78900). doi:10.1145/1273496.1273501 (https://doi.org/10.1145%2F1273496.1273501). ISBN 9781595937933.
3. Matthies, H.; Strang, G. (1979). "The solution of non linear finite element equations". *International Journal for Numerical Methods in Engineering*. **14** (11): 1613–1626. doi:10.1002/nme.1620141104 (https://doi.org/10.1002%2Fnme.1620141104).
4. Nocedal, J. (1980). "Updating Quasi-Newton Matrices with Limited Storage". *Mathematics of Computation*. **35** (151): 773–782. doi:10.1090/S0025-5718-1980-0572855-7 (https://doi.org/10.1090%2FS0025-5718-1980-0572855-7).
5. Byrd, R. H.; Nocedal, J.; Schnabel, R. B. (1994). "Representations of Quasi-Newton Matrices and their use in Limited Memory Methods". *Mathematical Programming*. **63** (4): 129–156. doi:10.1007/BF01582063 (https://doi.org/10.1007%2FBF01582063).
6. Byrd, R. H.; Lu, P.; Nocedal, J.; Zhu, C. (1995). "A Limited Memory Algorithm for Bound Constrained Optimization". *SIAM J. Sci. Comput.* **16** (5): 1190–1208. doi:10.1137/0916069 (https://doi.org/10.1137%2F0916069).
7. Zhu, C.; Byrd, Richard H.; Lu, Peihuang; Nocedal, Jorge (1997). "L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization". *ACM Transactions on Mathematical Software*. **23** (4): 550–560. doi:10.1145/279232.279236 (https://doi.org/10.1145%2F279232.279236).
8. N. Schraudolph, J. Yu, and S. Günter (2007). *A stochastic quasi-Newton method for online convex optimization*. AISTATS.
9. A. Mokhtari and A. Ribeiro (2015). "Global convergence of online limited memory BFGS" (http://www.jmlr.org/papers/volume16/mokhtari15a/mokhtari15a.pdf) (PDF). *Journal of Machine Learning Research*. **16**: 3151–3181.
10. A. Mokhtari and A. Ribeiro (2014). "RES: Regularized Stochastic BFGS Algorithm" (http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6899692&url=http%3A%2F%2Fieeexplore.ieee.org%2Fstamp%2Fstamp.jsp%3Ftp%3D%26arnumber%3D6899692). *IEEE Transactions on Signal Processing*. **62** (23): 6089–6104. doi:10.1109/TSP.2014.2357775 (https://doi.org/10.1109%2FTSP.2014.2357775).

11. Morales, J. L.; Nocedal, J. (2011). "Remark on "algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization" ". *ACM Transactions on Mathematical Software*. **38**: 1. doi:10.1145/2049662.2049669 (https://doi.org/10.1145%2F2049662.2049669).
12. "General-purpose Optimization" (http://finzi.psych.upenn.edu/R/library/stats/html/optim.html). *R documentation*. Comprehensive R Archive Network.

Further reading

- Liu, D. C.; Nocedal, J. (1989). "On the Limited Memory Method for Large Scale Optimization" (http://www.ece.northwestern.edu/~nocedal/PSfiles/limited-memory.ps.gz). *Mathematical Programming B*. **45** (3): 503–528. doi:10.1007/BF01589116 (https://doi.org/10.1007%2FBF01589116).
- Byrd, Richard H.; Lu, Peihuang; Nocedal, Jorge; Zhu, Ciyou (1995). "A Limited Memory Algorithm for Bound Constrained Optimization" (http://www.ece.northwestern.edu/~nocedal/PSfiles/limited.ps.gz). *SIAM Journal on Scientific and Statistical Computing*. **16** (5): 1190–1208. doi:10.1137/0916069 (https://doi.org/10.1137%2F0916069).
- Haghighi, Aria (2 Dec 2014). "Numerical Optimization: Understanding L-BFGS" (http://aria42.com/blog/2014/12/understanding-lbfgs).