OPT2021: 13th Annual Workshop on Optimization for Machine Learning

Faster Quasi-Newton Methods for Linear Composition Problems

Betty Shea SHEAWS@CS.UBC.CA
Mark Schmidt SCHMIDTM@CS.UBC.CA
University of British Columbia, Canada

Abstract

The limited memory version of BFGS (l-BFGS) is widely used for large-scale optimization problems despite having limited theoretical justification. In practice, l-BFGS tends to find solutions significantly faster than other methods with known convergence rates. Similarly, methods that use line searches outperform those that use constant stepsizes. Theory, however, usually shows the same worst-case complexity for both stepsize approaches. In this paper, we propose a practical modification to l-BFGS for linear composition problems. Our method combines l-BFGS with a momentum direction using efficient subspace optimization (SO). We extend the Wolfe conditions from one- to multi-dimension and experimentally compare our method to standard l-BFGS and to another SO method that is known to be efficient.

Keywords: subspace optimization, quasi-Newton, Wolfe conditions

1. Introduction

Quasi-Newton methods are known to work well in practice but do not have good theoretical guarantees. Until recently, BFGS [2, 6, 7, 24] and its limited memory version (l-BFGS [12]) are not known to converge meaningfully faster than standard gradient descent (GD). Rodomanov and Nesterov [20, 21, 22] showed recently that classical quasi-Newton methods have explicit local superlinear convergence rates. Kovalev et al. [9] showed that a randomized version of BFGS has local linear convergence for self-concordant functions and superlinear convergence with high probability for smooth and strongly convex functions. Fast-BFGS, a parallelizable version that combines BFGS with SO over a dynamic subspace, is shown to converge at least as fast as BFGS despite both lower storage requirements and lower per iteration cost [11]. Yet, despite these latest theoretical advances, standard BFGS and l-BFGS are popular mostly as a result of their practical importance.

Similarly, convergence analysis of iterative methods generally make the assumption that stepsizes are set exactly, or that a constant stepsize is used. Exact stepsizes, however, do not exist and linesearches generally outperform constant stepsizes. Furthermore, subspace optimization (SO), in other words setting stepsizes for more than one search direction, is relatively unexplored. Theory exist for SO but generally assumes that the search can be solved exactly. As in the case of a linesearch, this assumption is never satisfied in practice. In light of this, we explore practical considerations of inexact SO and extend Wolfe conditions to more than one dimension. We apply this to a method that combines a l-BFGS search direction with momentum. The inspiration comes from a known property of a subset of quasi-Newton methods that includes BFGS. When using a stepsize satisfying the Wolfe conditions, these methods exhibit local superlinear convergence [19]. Because we apply our technique to high-dimension datasets, we consider l-BFGS instead of BFGS. We find

that despite not having the same theoretical guarantees as BFGS, l-BFGS with SO performs very efficiently in practice.

1.1. Notation

A linear composition problem has an objective in the form

$$f(x) = F(Ax) \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$ is the feature matrix. The objective f is the composition of a linear map Ax with another function F. This is an important class of models in machine learning that include support vector machines (SVMs) and (binary or multiclass) logistic regression.

In this paper, we consider problems involving unconstrained minimization on a differentiable function f, or $\min_{x \in \mathbb{R}^n} f(x)$. We look at methods that arrive at a solution iteratively. Most iterative methods, including standard l-BFGS implementation, use a single direction to arrive at their next iterate. Instead, we assume that the method could use d directions. On the kth update,

$$x_{k+1} = x_k + P_k \alpha_k \tag{2}$$

where $P_k \in \mathbb{R}^{n \times d}$ contains d search directions as columns and $\alpha_k \in \mathbb{R}^d$ are non-negative stepsizes, or weights, for each of the directions. We denote the ith search direction as $P_{k,i}$ and refer to the first column $P_{k,1}$ as the primary direction. We also assume that there is a vector of stepsizes α_k that satisfy

$$(P_k \alpha_k)^\intercal \nabla f(x_k) < 0 \tag{3}$$

The intuition is that the average direction $P_k \alpha_k$ is a descent direction. Assumption (3) is guaranteed if we include a gradient direction or a quasi-Newton direction with a positive definite B_k as one of the columns of P_k. If negative stepsizes are allowed, we can relax (3) to

$$(P_k \alpha_k)^\intercal \nabla f(x_k) \neq 0$$

In the case where a method uses a single search direction, or $d = 1$, P_k is a column vector, α_k is a scalar and equation (2) reduces to the familiar form

$$x_{k+1} = x_k + \alpha_k P_k$$

When the SO subproblem at the kth iteration is solved inexactly, we start with some initial estimate of the stepsize that we denote α_k^0. We denote the differences in iterates as $s_k \triangleq x_{k+1} - x_k = P_k \alpha_k$ and the differences in gradients as $y_k \triangleq \nabla f(x_{k+1}) - \nabla f(x_k)$.

1.2. Quasi-Newton methods

Quasi-Newton methods use a search direction $P_{k,1} = -(B_k)^{-1} \nabla f(x_k)$ where positive definite matrix $B_k \in \mathbb{R}^{n \times n}$ approximates the Hessian $\nabla^2 f(x_k)$ through satisfying the secant equation

$$B_{k+1} s_k = y_k. \tag{4}$$

There are many variants of quasi-Newton methods and they differ in the way they update B_k. BFGS is a popular variant that uses a rank 2 update of B_k in the form

$$B_{k+1} = B_k - \frac{B_k s_k s_k^\intercal B_k}{s_k^\intercal B_k s_k} + \frac{y_k y_k^\intercal}{y_k^\intercal s_k}$$

l-BFGS reduces the storage requirements of B_k by storing instead a small number $c \ll n$ of vector pairs, s_k and y_k, that approximate B_k. These pairs are commonly referred to as *corrections*.

1.3. Momentum direction

In this paper, we focus on the case where there are two search directions, or $d = 2$, and where the second direction, $P_{k,2}$, is the momentum direction. In other words, the second search direction is the difference between the current and previous iterates, or $P_{k,2} = x_k - x_{k-1} = s_{k-1} = P_{k-1}\alpha_{k-1}$. There are many theoretical reasons for using momentum as a secondary direction [13, 18].

For linear composition problems, adding a momentum search direction does not increase the per iteration complexity and thus, is also useful from a practical point of view. After calculating the first candidate step size, any subsequent trial step size does not require any new matrix-vector multiplication. This is explained by equations (5) and (6) in sections 2.1 and 2.2 below.

1.4. Contribution

While inexpensive linesearches have been shown to work well with coordinate descent [17], this has not been adapted to quasi-Newton methods. For linear composition problems (1), SO has been shown to work well with truncated-Newton, GD and SVMs through a method called SESOP [14, 15, 25] but have not been used with quasi-Newton methods. Furthermore, our method uses inexact SO to set the weights (stepsizes) and is efficient for linear composition objectives. We show experimentally that our method performs better than both standard l-BFGS and SESOP. As far as we know, this paper is also the first to address practical issues around SO such as initialization and the choice of methods used to solve the SO subproblem.

2. Efficient stepsize searches

In large-scale applications that involve linear composition problems, the matrix A in equation (1) could be large. If F is a relatively simple calculation, then matrix-vector multiplications Av, for different vectors v, could be the bottleneck of the optimization problem. Thus, we want to minimize the number of Av operations we perform. For typical iterative methods, we might calculate two quantities at every iteration: the objective value $f(x)$ and the gradient $\nabla f(x)$. The structure of linear composition problems allow us to minimize the number of matrix-vector multiplications for each of the three quantities.

2.1. Linesearch

Consider the case where the next iterate is determined along a single search direction $P_{k,1} = p_k$. At each iteration, given p_k, the function value at the next iterate

$$f(x_{k+1}) = F(A(x_k + \alpha_k p_k)) = F(Ax_k + \alpha_k Ap_k) = f(v_k + \alpha_k p'_k) \tag{5}$$

where the values $v_k = Ax_k$ and $p'_k = Ap_k$ are vectors that can be stored and reused. Thus, the function value of $f(x_{k+1})$ could be calculated with only two matrix-vector multiplications and several calls to f for many α_k values. In other words, after the first iteration, every subsequent iteration of the linesearch requires only a scalar-vector multiplication rather than a matrix-vector multiplication. This trick is widely used for coordinate descent but have not been used previously for quasi-Newton methods.

Step size selection	Mat-vec multiplications	Count
Fixed $1/L$	$\nabla f = A^\intercal \nabla F$, $A\nabla f$	2
Line search, e.g. Armijo, Wolfe	as above	2
Plane search with momentum direction	as above	2
Plane search with generic direction	as above and Ap_2	3

Table 1: Number of matrix-vector multiplications per iteration of l-BFGS or GD for objectives of the form $f(x) = F(Ax)$ and second direction p_2 in the case of a plane search. There is an additional matrix-vector multiplication at the first iteration for Ax_0. Subsequent calculations of Ax_k do not require matrix-vector multiplications using (5).

2.2. Subspace search

Linesearches are used everywhere in practice but SO is less commonly employed due to cost. Nevertheless, SO has also been shown to be efficient for objectives that are compositions of an expensive linear map and a cheaper non-linear function by Narkiss and Zibulevsky [14]. The authors also showed optimal $O(1/k^2)$ convergence rates on convex objectives that include the current gradient and weighted previous gradients as search directions. The convergence properties, however, rely on being able to solve the SO subproblem exactly.

Analogous to equation (5), when $d > 1$, inexact SO is also efficient because

$$f(x_{k+1}) = F\left(A\left(x_k + P_k\alpha_k\right)\right) = F(Ax_k + AP_k\alpha_k) = f(v_k + P'_k\alpha_k) \tag{6}$$

where $P'_k = AP_k$ is also stored and reused. For a P_k with d generic directions, computing AP_k requires d matrix-vector multiplications. Thus, going from a line search to a two-dimensional plane search would result in one additional matrix-vector multiplication. Adding the most recent momentum direction, however, is a special case where there are no additional cost. Calculating AP_k is the same as calculating Ax_k and $A(x_k - x_{k-1})$. If we store Ax_{k-1} from the previous iteration, we can perform a two-dimensional subspace search for the same cost as a one-dimensional linesearch. This observation is illustrated in Table 1 using GD as the primary direction.

2.2.1. MULTI-DIMENSION WOLFE CONDITIONS

The Wolfe conditions are a popular method for performing an inexact linesearch. Here, we directly extend the Wolfe conditions from one- to multi-dimension. This is given by the sufficient decrease (or Armijo) condition

$$f(x_k + P_k\alpha_k) \leq f(x_k) + c_1\nabla f(x_k)^\intercal(P_k\alpha_k) \tag{7}$$

and the curvature condition

$$\nabla f(x_k + P_k\alpha_k)^\intercal(P_k\alpha_k) \geq c_2\nabla f(x_k)^\intercal(P_k\alpha_k) \tag{8}$$

where $c_1 \in (0, \frac{1}{2})$ and $c_2 \in (c_1, 1)$ are constants. A similar equation to (7) was previously used in Conn et al. [4].

Compared to the one-dimension version, the conditions on the next iterate is now a function of the average direction $P_k\alpha_k$ instead of a single direction p_k. The underlying intuition remains the same. Equation (7) ensures that the next iterate improves on the objective. Equation (8) ensures that the magnitude of the directional derivative cannot be too large and thus the new iterate is an approximate stationary point along the average direction.

Given a positive definite estimate of the Hessian $B_k \approx \nabla^2 f(x_k)$, quasi-Newton methods satisfy the secant equation (4). This requires that B_{k+1} maps s_k to y_k and is satisfied when $y_k^\intercal s_k > 0$. For strongly convex functions, this condition is automatically satisfied. For nonconvex function, the curvature inequality in the Wolfe conditions guarantee that this is satisfied. This property is retained in the multi-dimension version (8) if assumption (3) is satisfied. For details, see Appendix A.

2.3. Practical SO issues

The performance of inexact stepsize search methods rely greatly on practical issues. While these issues have largely been explored in the case of a linesearch (for example, see chapter 3 of Nocedal and Wright [16]), there are fewer guidelines in the case of SO.

2.3.1. INITIALIZATION

We discuss two ways to set the initial trial stepsize at every iteration.

Newton step One of the conditions for local superlinear convergence of quasi-Newton methods is to always initialize the trial stepsize to 1. For example, if the primary direction is BFGS, we should set $\alpha_k^0 = \begin{bmatrix}1 & 0 & \dots & 0\end{bmatrix}^\intercal$. Because its iterates do not converge to Newton iterates, there is little theoretical reason to use Newton step initialization for l-BFGS. Yet Newton step initialization works well for l-BFGS in practice and is often the default initialization method in optimization code.

Linesearch initialization Another way to set α_k^0 is to first run a linesearch on the primary direction to obtain a satisfying stepsize β. We can then set $\alpha_k^0 = \begin{bmatrix}\beta & 0 & \dots & 0\end{bmatrix}^\intercal$. If the primary direction comes from a method that is known to converge with a linesearch, this initialization method guarantees convergence of the overall method.

2.3.2. CHOICE OF SO METHOD

Experimentally, we found that using Barzilai-Borwein (BB) [1] as the method to solve the SO subproblem works significantly better than using GD. BB is known to have fast convergence for quadratic functions.

3. Experiments

We compared our modified l-BFGS method (`lBFGS-SO`) against three different benchmarks: (a) standard l-BFGS using Wolfe conditions (`lBFGS-W-def`), (b) l-BFGS with a linesearch optimized for linear composition problems (`lBFGS-W-opt`) and (c) a method that combines GD with a momentum term using subspace optimization that is optimized for linear composition problems (`GD-SO`).

The optimization software developed for the experiments is inspired by `minFunc` [23]. The code for these experiments could be found in the `minFuncSO` repository.[1] Our main experiments

1. https://github.com/sheaws/minFuncSO.

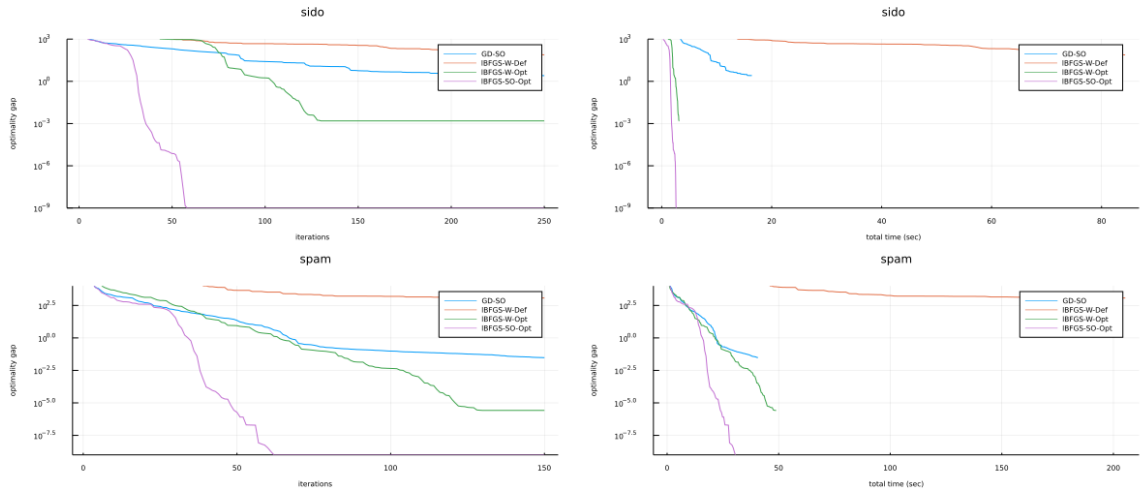

Figure 1: Binary classification of the *sido* and *spam* datasets. Left plot shows accuracy by number of iterations and right plot shows accuracy by time taken in seconds.

are run on two real datasets *sido* and *spam*. The *sido* dataset comes from the Causality Workbench website [8] and contains 12,678 datapoints and 4,932 variables. The *spam* dataset was prepared by Carbonetto [3] using the TREC 2005 corpus [5] and contains 92,189 datapoints and 823,470 variables. The task is binary classification using logistic regression. `lBFGS-SO` and `GD-SO` both use one additional direction that is the most recent momentum term and the SO is solved using BB. We plot the results of our experiments in figure 1.

Our experiments suggest that, in the case of logistic regression on linear composition problems, there are significant gains from adding the most recent momentum term as a secondary search direction and using SO. In figure 1, our method (in purple) performs better than other methods measured in terms of accuracy, number of iterations and time taken.

Finally, we performed additional experiments on 49 binary classification datasets made available through Dataset Downloader [10]. Preliminary results suggest that SO may work particularly well in the overparameterized setting. On the other hand, it generally does not hurt to go from a line search to a subspace search. Further details of the experiments are given in Appendix B.

4. Discussion

In this paper, we showed that for linear composition problems, our modified version of l-BFGS may find more accurate solutions in less time. In particular, this method seem to perform better in the overparameterized setting. More analytical work needs to be done to determine the conditions where it helps to go from a line search to a subspace search.

Beyond linear composition problems, there are other important problems commonly found in machine learning that could also benefit from this modified version of l-BFGS. These problems fall under a class of problems known as multilinear composition problems, which have optimization objectives that are multilinear maps. This broad category includes familiar problems such as matrix

completion and factorization problems and optimizing the log-determinant of a matrix. It also encompasses the linear composition problem explored in this paper.

References

[1] Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8:141–148, 1988.

[2] C.G. Broyden. Quasi-newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, 1967.

[3] Peter Carbonetto. *New Probabilistic Inference Algorithms that Harness the Strengths of Variational and Monte Carlo Methods. PhD thesis, University of British Columbia.*, 2009.

[4] A.R. Conn, Nick Gould, A. Sartenaer, and Ph. L. Toint. On iterated-subspace minimization methods for nonlinear optimization. *AMS-IMS-SIAM*, pages 50–78, 2018.

[5] Gordon V. Cormack and Thomas R. Lynam. *Spam Corpus Creation for TREC*, 2005. URL https://plg.uwaterloo.ca/~gvcormac/treccorpus/.

[6] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3): 317–323, 1970.

[7] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[8] Isabelle Guyon. Sido: A pharmacology dataset. 2008. URL http://www.causality.inf.ethz.ch/data/SIDO.html.

[9] Dmitry Kovalev, Robert M. Gower, Peter Richtárik, and Alexander Rogozin. Fast linear convergence of randomized bfgs. 2021. URL https://arxiv.org/pdf/2002.11337.pdf.

[10] Frederik Kunstner. Dataset downloader. 2021. URL https://github.com/fKunstner/dataset-downloader.

[11] Zheng Li, Shi Shu, and Jian-Ping Zhang. A dynamic subspace based bfgs method for large scale optimization. 2020. URL https://arxiv.org/abs/2001.07335.

[12] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[13] A. Miele and J.W. Cantrell. Study on a memory gradient method for the minimization of functions. *Journal of Optimization Theory and Applications*, 3(6), 1969.

[14] Guy Narkiss and Michael Zibulevsky. Sequential subspace optimization method for large-scale unconstrained problems. Technical report, Technion - Israel Institute of Technology, 2005.

[15] Guy Narkiss and Michael Zibulevsky. Support vector machine via sequential subspace optimization. Technical report, Technion - Israel Institute of Technology, 2005.

[16] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization, 2nd Ed.* Springer, 2006.

[17] Julie Nutini, Issam Laradji, and Mark Schmidt. Let's make block coordinate descent go fast: Faster greedy rules, message-passing, active-set complexity, and superlinear convergence. 2017. URL https://arxiv.org/abs/1712.08859.

[18] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[19] M.J.D. Powell. On the convergence of the variable metric algorithm. *IMA Journal of Applied Mathematics*, 7(1):21–36, 1971.

[20] Anton Rodomanov and Yurii Nesterov. Greedy quasi-newton methods with explicit superlinear convergence. *SIAM J. Optim*, 31(1):785–811, 2021.

[21] Anton Rodomanov and Yurii Nesterov. Rates of superlinear convergence for classical quasi-newton methods. *Mathematical Programming*, 2021.

[22] Anton Rodomanov and Yurii Nesterov. New reesults on superlinear convergence of classical quasi-newton methods. *Journal of Optimization Theory and Applications*, 188:744–769, 2021.

[23] Mark Schmidt. minfunc: unconstrained differentiable multivariate optimization in matlab. 2005. URL https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html.

[24] D.F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[25] Michael Zibulevsky. Sesop-tn: Combining sequential subspace optimization with truncated newton method. 2008. URL https://ie.technion.ac.il/~mcib/sesoptn_paper1.pdf.

Appendix A. Multi-dimension Wolfe and the secant equation

The Wolfe conditions guarantee that the secant equation is satisfied. In other words, for B_k to map the change in iterates s_k to the change in gradients y_k, it must be the case that $y_k^\intercal s_k > 0$. Here we show that multi-dimension Wolfe retains this property.

From equation (8)

$$\nabla f(x_{k+1})^\intercal (P_k \alpha_k) \geq c_2 \nabla f(x_k)^\intercal (P_k \alpha_k)$$

Subtract $\nabla f(x_k)^\intercal (P_k \alpha_k)$ from both sides

$$[\nabla f(x_k + 1) - \nabla f(x_k)]^\intercal (P_k \alpha_k) \geq [(c_2 - 1) \nabla f(x_k)]^\intercal (P_k \alpha_k)$$

Rewrite

$$y_k^\intercal s_k \geq (c_2 - 1) \nabla f(x_k)^\intercal (P_k \alpha_k)$$

Because $c_2 \in (c_1, 1)$, $c_2 - 1 < 0$. Assumption (3) means that $\nabla f(x_k)^\intercal (P_k \alpha_k) < 0$. Therefore, the righthand side in the above inequality is strictly larger than 0 and

$$y_k^\intercal s_k > 0$$

Appendix B. Further details for experiments

Values of the experiments using datasets *sido* and *spam* are given in table 2. We also performed the same comparisons for 49 additional binary classification datasets. The results are summarized in tables 3 (for datasets where logistic regression perfectly separates the two classes) and 4 (for the remaining datasets). SO appears to work particularly well on perfectly separable datasets. A typical convergence plot looks like the one for the leukemia dataset give in figure 2. For datasets that are not separable, a linesearch could be faster in time. For example, the convergence plot of the w7a dataset is given in figure 3.

sido	Method	$f(x_*)$	Time (sec)	Outer Iters	Inner Iters	Mat-Vec
	GD-SO	2.516832407	21.7	250	21,791	7,423
	l-BFGS-Wolfe-default	75.74420322	85.1	250	1,375	4,054
	l-BFGS-Wolfe-optimized	0.001548940	3.2	130	805	262
	l-BFGS-SO	0.000000017	2.7	57	2,777	1,729
spam	Method	$f(x_*)$	Time (sec)	Outer Iters	Inner Iters	Mat-Vec
	GD-SO	0.030161545	40.6	150	6,695	4,791
	l-BFGS-Wolfe-default	1176.191233	205.1	150	969	2,814
	l-BFGS-Wolfe-optimized	0.000003176	47.0	130	829	262
	l-BFGS-SO	0.000000549	27.8	61	1,973	1,675

Table 2: Binary classification of the *sido* and *spam* datasets.

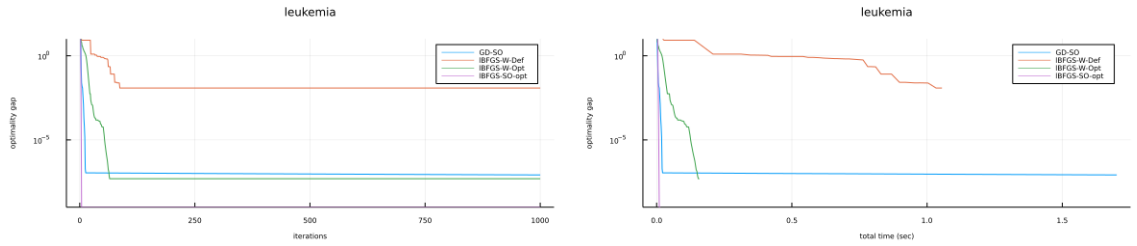

Figure 2: Convergence plots using the leukemia dataset. Left plot shows accuracy by number of iterations and right plot shows accuracy by time taken in seconds.

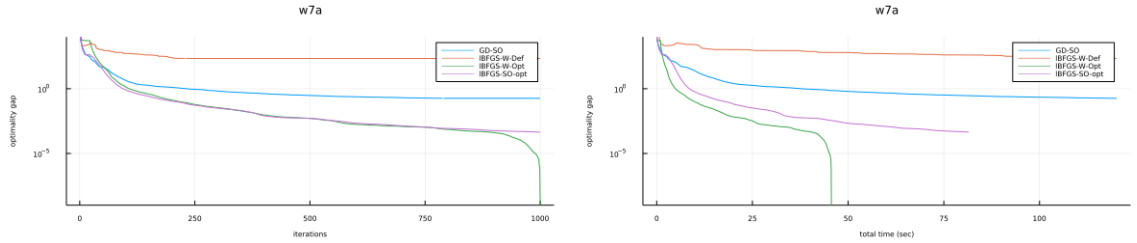

Figure 3: Convergence plots using the w7a dataset. Left plot shows accuracy by number of iterations and right plot shows accuracy by time taken in seconds.

Dataset	m	n	$f(x_*)$	Best Method
breast-cancer	683	10	0.000	l-BFGS-Wolfe-optimized
breast-cancer_scale	683	10	0.000	l-BFGS-Wolfe-optimized
colon-cancer	62	2,000	0.000	l-BFGS-SO
covtype.binary	581,012	54	0.000	l-BFGS-SO
covtype.binary.scale	581,012	54	0.000	l-BFGS-SO
duke-breast-cancer	38	7,129	0.000	l-BFGS-SO
gisette	1,000	5,000	0.000	l-BFGS-SO
leukemia	34	7,129	0.000	l-BFGS-SO
madelon	600	500	0.000	l-BFGS-SO
mushrooms	8,124	112	0.000	l-BFGS-SO
svmguide3	41	22	0.000	l-BFGS-SO

Table 3: Binary classification of 11 separable datasets. m is the number of samples. n is the dimensionality of the problem. Of the four methods, l-BFGS combined with a line search that satisfies the Wolfe conditions (optimized for linear composition problems) performed best on two of the 11 datasets; whereas l-BFGS combined with momentum and SO performed best on 9 out of the 11 datasets.

Dataset	m	n	$f(x_*)$	Best Method
a1a	30,956	123	9,965.230	l-BFGS-SO
a2a	30,296	123	9,698.535	l-BFGS-Wolfe-optimized
a3a	29,376	123	9,432.894	l-BFGS-Wolfe-optimized
a4a	27,780	123	8,916.993	l-BFGS-Wolfe-optimized
a5a	26,147	123	8,402.151	l-BFGS-Wolfe-optimized
a6a	21,341	123	6,820.987	l-BFGS-Wolfe-optimized
a7a	16,461	123	5,208.236	l-BFGS-SO
a8a	9,865	122	3,038.297	l-BFGS-Wolfe-optimized
a9a	16,281	122	5,188.674	l-BFGS-Wolfe-optimized
australian	690	14	351.958	GD-SO
australian_scale	690	14	221.780	l-BFGS-Wolfe-optimized
cod-rna	157,413	8	20,086.953	l-BFGS-SO
diabetes	768	8	467.326	l-BFGS-Wolfe-optimized
diabetes_scale	768	8	361.823	l-BFGS-Wolfe-optimized
fourclass	862	2	459.449	l-BFGS-SO
fourclass_scale	862	2	456.814	l-BFGS-SO
german.numer	1,000	24	471.626	l-BFGS-SO
german.numer_scale	1,000	24	468.417	l-BFGS-Wolfe-optimized
heart	270	13	93.814	l-BFGS-SO
heart_scale	270	13	95.082	l-BFGS-SO
ijcnn1	91,701	22	17,935.317	l-BFGS-Wolfe-optimized
ionosphere	351	34	102.134	l-BFGS-Wolfe-optimized
news20.binary	19,996	1,355,191	101.473	l-BFGS-SO
phishing	11,055	68	3,395.035	l-BFGS-SO
rcv1.binary	677,399	47,236	50,321.757	GD-SO
real-sim	72,309	20,958	236.62	l-BFGS-Wolfe-optimized
skin_nonskin	245,057	3	177.446	l-BFGS-Wolfe-default
splice	2,175	60	682.464	l-BFGS-Wolfe-optimized
sonar	208	60	30.010	l-BFGS-Wolfe-optimized
svmguide1	4,000	4	1,386.294	l-BFGS-Wolfe-default
w1a	47,272	300	5,212.191	GD-SO
w2a	46,279	300	5,089.004	l-BFGS-Wolfe-optimized
w3a	44,837	300	4,942.815	l-BFGS-Wolfe-optimized
w4a	42,383	300	4,670.944	l-BFGS-Wolfe-optimized
w5a	39,861	300	4,339.669	l-BFGS-SO
w6a	32,561	300	3,525.112	l-BFGS-SO
w7a	25,057	300	2,687.469	l-BFGS-Wolfe-optimized
w8a	14,951	300	1,521.792	l-BFGS-Wolfe-optimized

Table 4: Binary classification of 38 datasets that are not separable. m is the number of samples. n is the dimensionality of the problem. Gradient descent combined with momentum and SO performed best for 3 datasets; l-BFGS combined with a line search satisfying Wolfe conditions (non-optimized for linear composition problems) performed best on 2 datasets; l-BFGS combined with a line search satisfying Wolfe conditions (optimized) performed best for 21 datasets; l-BFGS combined with momentum and SO performed best on 12 datasets.