Project Report (795): Newton's Method Quasi-Newton methods (Literature Survey)

Pankaj Kumar Jatav – G10338769

May 5, 2022

Abstract

Machine learning is developing rapidly, has made many theoretical breakthroughs and is widely applied in many fields. Optimization, which is an important part of machine learning, has attracted much attention from researchers. With the exponential growth in the amount of data and the increasing complexity of models, optimization methods in machine learning are increasingly facing more and more challenges. Many works on solving optimization problems or improving optimization methods in machine learning have been proposed consecutively. It is of great importance to systematically review and summarize optimization methods from a machine learning standpoint, which can provide guidance for developments in machine learning research and optimization. And, Newton's method is a fundamental tool in numerical analysis and a wide range of applications, such as operations research and data mining. We look at the method's history, main ideas, convergence results, modifications, and overall behavior. We look at how the method can be used to solve unconstrained minimization, equality constrained problems, convex programming, and interior point methods, among other types of optimization problems. Some expansions (non-smooth issues, continuous analog, Smale's results, and so on) are briefly treated, while others (e.g., variations of the global convergence approach) are discussed in greater depth.

1 Introduction

Recently machine learning has been grown at a remarkable rate and this field has been keep growing more and more. Many researchers have been attracted to machine learning. Machine learning play a significant role in the image processing, audio/video processing, machine translation and many more. Optimization is one of the core components of machine learning. The heart of the machine leaning to build an optimized model and finding pattern on given data and learn these patterns to apply for unseen data. In the age of massive data, the efficacy and efficiency of numerical optimization techniques have a significant impact on the popularity and implementation of optimize machine learning models. On the basis of gradient information in optimization in machine learning, most popular optimization methods can be divided into three categories: First order optimizations methods, which are widely used as stochastic gradient methods; Second order optimizations methods, in which Newton's methods is a mostly used example; Derivative free optimization methods, in which Coordinate descent method is typical example. We are exploring the newton's methods and it's different variant in this report.

One of the most important tools in numerical analysis, operations research, optimization, and control is Newton's method. It can be used in management science, industrial and financial research, and data mining, among other things. Its importance in optimization cannot be overstated: the method provides the foundation for the most effective linear and nonlinear programming procedures. Newton's approach, for example, is used in polynomial timeinterior point algorithms in convex optimization. The current paper examines the method's fundamental concepts from a historical viewpoint, as well as current research and applications. Our purpose is to address the most difficult issues in this field of research and to provide appropriate references for further study.

We begin by addressing the one dimension case. While it does not address all of the concerns, it does emphasize the essential issue of local convergence as the best option. Then we will discuss other improved methods based on the newton's methods which are quasi-newton methods and it's variant.

2 Newton's Method

2.1 One dimensional case:

Simply use the section and subsection commands, as in this example document! With Overleaf, all the formatting and numbering is handled automatically according to the template you've chosen. If you're using Rich Text mode, you can also create new section and subsections via the buttons in the editor toolbar.

$$x_{t+1} := x_t - \frac{f(x_t}{f'(x_t)}, t \geq 0$$

Below Figure shows what happens. x_{t+1} is the point where the tangent line to the graph of f at $(x_t\ ,\ f(x_t))$ intersects the x-axis. In formulas, x_{t+1} is the solution of the linear equation.

$$f(x_t) + f'(x_t)(x - x_t) = 0$$

and this yields the update formula from equation.

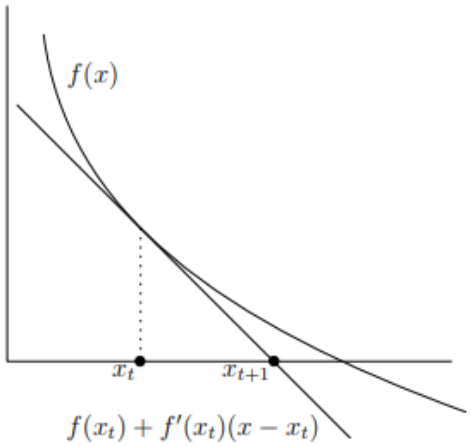

Figure 1: One step on newton's method

2.2 Newton's method for optimization

Suppose we want to find a global minimum x_* of a differentiable convex function f : R → R (assuming that a global minimum exists). To do this, we can apply Newton's method if f is twice differentiable; the update step then becomes

$$x_{t+1} := x_t - \frac{f'(x_t}{f''(x_t)}, t \geq 0$$

$$x_{t+1} := x_t - f''(x_t)^{-1} f'(x_t), t \geq 0$$

There's no reason to confine to d = 1. Here is Newton's strategy for minimizing a raised work f : R^d → R. We select x0 self-assertively and then iterate:

$$x_{t+1} := x_t - f''(x_t)^{-1} f'(x_t), t \geq 0$$

The update vector $f''(x_t)^{-1} f'(x_t)$ is the result of a matrix-vector multiplication: we invert the Hessian at x_t and multiply the result with the gradient at x_t. As before, this fails if the Hessian is not invertible, and may get out of control if the Hessian has small norm.

We can consider the a special case of a general update schema

$$x_{t+1} := x_t - H(x_t)f'(x_t), t \geq 0$$

where $H(x_t) \in \mathrm{R}^{dxd}$ is some matrix, then we can also see the above equation in gradient descent in this form also also where $H(x_t)$ is αI. Where alpha is a learning rate. We will also see the different $H(x_t)$ functions soon.

2.3 Multivariate Newton's Method Update

- Recall the inverse of a matrix , written , is the matrix such that , the identity matrix
- Inverses do not always exist, if the inverse doesn't exist, then there may be no solution or infinitely many solutions
- Computing the inverse can be expensive: requires operations for an matrix
- Computing the Hessian matrix itself can be computationally expensive
- Can converge faster than gradient methods, but is less robust in general

3 Quasi-Newton Methods

The main computational bottleneck in Newton's method is the computation and inversion of the Hessian matrix in each step. This matrix has size d × d, so it will take up to $O(d^3)$ time to invert it. Already in the 1950s, attempts were made to circumvent this costly step, the first one going back to Davidon [Dav59]. Now we will (for a change) not prove convergence results; rather, we focus on the development of Quasi-Newton methods, and how state-of-the-art methods arise from first principles. To motivate the approach, let us go back to the 1-dimensional case.

3.1 The secant method

Like Newton's strategy, the secant strategy is an iterative strategy for finding a zero of a univariate work. Not at all like Newton's strategy, it does not utilize subordinates and subsequently does not require the work beneath thought to be differentiable. In reality, it is (hence) much more seasoned than Newton's strategy. Turning around history and beginning from the Newton step

$$x_{t+1} := x_t - H(x_t)f'(x_t), t \geq 0$$

we are able determine the secant strategy by supplanting the subordinate $f'(x_t)$ with its finite contrast guess

$$\frac{f(x_t) - f(x_{t-1}}{x_t - x_{t-1}}$$

As we have

$$f'(x_t) = lim_{x \to x_t} \frac{f(x_t) - f(x)}{x_t - x}$$

for $|x_t - x_{t-1}|$ small. As the methods proceeds, we expect consecutive iterates x_{x-1}, x_t to become closer and closer, so that the secant step

$$x_{t+1} := x_t - f(x_t)\frac{x_t - x_{t-1}}{f(x_t) - f(x_{t-1})}, t \geq 1$$

When the assignment is to optimize a differentiable univariate work, we can apply the secant strategy to its subordinate to get the secant method for optimization:

$$x_{t+1} := x_t - f'(x_t)\frac{x_t - x_{t-1}}{f'(x_t) - f'(x_{t-1})}, t \geq 1$$

This is equivalent to newton's methods for optimization. Where newton methods use the second derivative but here it is second derivative free.

3.2 The secant condition

Applying limited distinction guess to the moment subsidiary of f (we're still within the 1-dimensional case), we get

$$H_t := \frac{x_t - x_{t-1}}{f'(x_t) - f'(x_{t-1})} \approx f''(x_t)$$

Can also we written as:

$$f'(x_t) - f'(x_{t-1}) = H_t(x_t - x_{t-1}) \approx f''(x_t)(x_t - x_{t-1}$$

As we the know that newton methods for optimization uses below step:

$$x_{t+1} := x_t - f''(x_t)^{-1} f'(x_t), t \geq 0$$

The secant method works with the approximation $H_t \approx f''(x_t)$

$$x_{t+1} = x_t - H_t^{-1} f'(x_t)$$

The reality that Ht approximates $f''(x_t)$ within the twice differentiable case was our inspiration for the secant strategy, but within the strategy itself, there is no reference to f'' (which is precisely the point). All that's required is the secant condition that characterizes H_t :

$$f'(x_t) - f'(x_{t-1}) = H_t(x_t - x_{t-1})$$

3.3 Quasi-Newton methods

In the event that f is twice differentiable, the secant condition along side the first order Taylor estimation of f'(x) yields the d-dimensional analog of

$$f'(x_t) - f'(x_{t-1} = H_t(x_t - x_{t-1} \approx f''(x_t)(x_t - x_{t-1})$$

We might hence trust that $H_t0"(x_t)$, and this would cruel that approximates Newton's strategy. In this manner, at whatever point we utilize with a symmetric lattice fulfilling the secant condition, we say that we have a Quasi-Newton strategy.

Within the 1-dimensional case, there's as it were one Quasi-Newton method— the secant strategy. Undoubtedly, condition extraordinarily characterizes the number H_t in each step.

But within the d-dimensional case, the framework Ht in the secant condition is underdetermined, beginning from d = 2: Taking the symmetry requirement into account, may be a framework of d conditions in d(d + 1)/2 questions, so if it is satisfiable at all, there are numerous arrangements Ht . This raises the question of which one to select, and how to do so effectively; after all, we need to get a few investment funds over Newton's strategy.

Newton's strategy could be a Quasi-Newton method if and as it were in case f may be a nondegenerate quadratic work (Work out 54). Consequently, Quasi-Newton strategies don't generalize Newton's strategy but frame a family of related calculations.

The primary Quasi-Newton strategy was created by William C. Davidon in 1956; he frantically required cycles that were speedier than those of Newton's strategy in order get comes about within the brief time ranges between anticipated disappointments of the room-sized computer that he utilized to run his computations on.

But the paper he composed almost his modern strategy got rejected for lacking a merging investigation, and for purportedly questionable documentation. It got to be a very compelling Specialized Report in 1959 and was at last officially published in 1991, with a foreword giving the verifiable setting. Ironically, Quasi-Newton strategies are nowadays the strategies of choice in a number of pertinent machine learning applications.

3.4 BFGS Algorithm

The algorithm begins at an initial estimate for the optimal value $\mathbf{x}_0$ and proceeds iteratively to get a better estimate at each stage.

The search direction pk at stage k is given by the solution of the analogue of the Newton equation:

$$B_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k),$$

where $B_k B_k$ is an approximation to the Hessian matrix, which is updated iteratively at each stage, and $\nabla f(\mathbf{x}_k)\nabla f(\mathbf{x}_k)$ is the gradient of the function evaluated at x_k. A line search in the direction p_k is then used to find the next point x_{k+1} by minimizing $f(\mathbf{x}_k + \gamma \mathbf{p}_k)$ over the scalar $\gamma > 0.\gamma > 0$.

The quasi-Newton condition imposed on the update of $B_k B_k$ is

$$B_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

Let $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, then $B_{k+1}B_{k+1}$ satisfies $B_{k+1}\mathbf{s}_k = \mathbf{y}_k B_{k+1}\mathbf{s}_k = \mathbf{y}_k$, which is the secant equation. The curvature condition $\mathbf{s}_k^\top \mathbf{y}_k > 0\mathbf{s}_k^\top \mathbf{y}_k > 0$ should be satisfied for $B_{k+1}B_{k+1}$ to be positive definite, which can be verified by pre-multiplying the secant equation with $\mathbf{s}_k^T \mathbf{s}_k^T$. If the function is not strongly convex, then the condition has to be enforced explicitly.

Instead of requiring the full Hessian matrix at the point $\mathbf{x}_{k+1}\mathbf{x}_{k+1}$ to be computed as $B_{k+1}B_{k+1}$, the approximate Hessian at stage k is updated by the addition of two matrices:

$$B_{k+1} = B_k + U_k + V_k.$$

Both $U_k U_k$ and $V_k V_k$ are symmetric rank-one matrices, but their sum is a rank-two update matrix. BFGS and DFP updating matrix both differ from its predecessor by a rank-two matrix. Another simpler rank-one method is known as symmetric rank-one method, which does not guarantee the positive definiteness. In order to maintain the symmetry and positive definiteness of $B_{k+1}B_{k+1}$, the update form can be chosen as $B_{k+1} = B_k + \alpha \mathbf{u}\mathbf{u}^\top + \beta \mathbf{v}\mathbf{v}^\top B_{k+1} = B_k + \alpha \mathbf{u}\mathbf{u}^\top + \beta \mathbf{v}\mathbf{v}^\top$. Imposing the secant condition, $B_{k+1}\mathbf{s}_k = \mathbf{y}_k B_{k+1}\mathbf{s}_k = \mathbf{y}_k$. Choosing $\mathbf{u} = \mathbf{y}_k \mathbf{u} = \mathbf{y}_k$ and $\mathbf{v} = B_k \mathbf{s}_k \mathbf{v} = B_k \mathbf{s}_k$, we can obtain:

$$\alpha = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k},$$

$$\beta = -\frac{1}{\mathbf{s}_k^T B_k \mathbf{s}_k}.$$

$\beta = -\dfrac{1}{\mathbf{s}_k^T B_k \mathbf{s}_k}$. Finally, we substitute α and β into $B_{k+1} = B_k + \alpha \mathbf{u}\mathbf{u}^\top + \beta \mathbf{v}\mathbf{v}^\top$ and get the update equation of B_{k+1}:

$$B_{k+1} = B_k + \frac{\mathbf{y}_k \mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}} \mathbf{s}_k} - \frac{B_k \mathbf{s}_k \mathbf{s}_k^{\mathrm{T}} B_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}} B_k \mathbf{s}_k}.$$

Algorithm:

From an initial guess $\mathbf{x}_0$ and an approximate Hessian matrix B_0 the following steps are repeated as $\mathbf{x}_k$ converges to the solution:

1. Obtain a direction $\mathbf{p}_k$ by solving $B_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$.
2. Perform a one-dimensional optimization (line search) to find an acceptable stepsize α_k in the direction found in the first step. If an exact line search is performed, then $\alpha_k = \arg\min f(\mathbf{x}_k + \alpha \mathbf{p}_k)$. In practice, an inexact line search usually suffices, with an acceptable α_k satisfying Wolfe conditions.
3. Set $\mathbf{s}_k = \alpha_k \mathbf{p}_k$ and update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$.
4. $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$.
5. $B_{k+1} = B_k + \dfrac{\mathbf{y}_k \mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}} \mathbf{s}_k} - \dfrac{B_k \mathbf{s}_k \mathbf{s}_k^{\mathrm{T}} B_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}} B_k \mathbf{s}_k}$.

3.5 L-BFGS 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 $f(\mathbf{x})f(\mathbf{x})$.

L-BFGS shares many features with other quasi-Newton algorithms, but is very different in how the matrix-vector multiplication $d_k = -H_k g_k$ is carried out, where d_k is the approximate Newton's direction, g_k is the current gradient, 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.

We 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

$$s_k = x_{k+1} - x_k$$

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

We define $\rho_k = \dfrac{1}{y_k^\top s_k}$, 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 ascent direction.

Thus we can compute the descent direction as follows:

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

This formulation gives the search direction for the minimization problem, i.e., $z = -H_k g_k$. For maximization problems, one should thus take -z instead. Note that the initial approximate inverse Hessian H_k^0 is chosen as a diagonal matrix or even a multiple of the identity matrix since this is numerically efficient.

The scaling of the initial matrix γ_k 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 $y_k^\top s_k > 0$ will be satisfied by the chosen step since a step length greater than $1 1 may be needed to satisfy this condition. Some implementations s$ 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.

3.6 The most popular update formulas are:

Method	$B_{k+1} =$	$H_{k+1} = B_{k+1}^{-1} =$
BFGS	$B_k + \frac{y_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k} - \frac{B_k \Delta x_k (B_k \Delta x_k)^{\mathrm{T}}}{\Delta x_k^{\mathrm{T}} B_k \, \Delta x_k}$	$\left(I - \frac{\Delta x_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k}\right) H_k \left(I - \frac{y_k \Delta x_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \Delta x_k}\right) + \frac{\Delta x_k \Delta x_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \, \Delta x_k}$
Broyden	$B_k + \frac{y_k - B_k \Delta x_k}{\Delta x_k^{\mathrm{T}} \, \Delta x_k} \Delta x_k^{\mathrm{T}}$	$H_k + \frac{(\Delta x_k - H_k y_k) \Delta x_k^{\mathrm{T}} H_k}{\Delta x_k^{\mathrm{T}} H_k \, y_k}$
Broyden family	$(1-\varphi_k) B_{k+1}^{\mathrm{BFGS}} + \varphi_k B_{k+1}^{\mathrm{DFP}}, \quad \varphi \in [0,1]$	
DFP	$\left(I - \frac{y_k \, \Delta x_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \, \Delta x_k}\right) B_k \left(I - \frac{\Delta x_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \, \Delta x_k}\right) + \frac{y_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} \, \Delta x_k}$	$H_k + \frac{\Delta x_k \Delta x_k^{\mathrm{T}}}{\Delta x_k^{\mathrm{T}} \, y_k} - \frac{H_k y_k y_k^{\mathrm{T}} H_k}{y_k^{\mathrm{T}} H_k y_k}$
SR1	$B_k + \frac{(y_k - B_k \, \Delta x_k)(y_k - B_k \, \Delta x_k)^{\mathrm{T}}}{(y_k - B_k \, \Delta x_k)^{\mathrm{T}} \, \Delta x_k}$	$H_k + \frac{(\Delta x_k - H_k y_k)(\Delta x_k - H_k y_k)^{\mathrm{T}}}{(\Delta x_k - H_k y_k)^{\mathrm{T}} y_k}$

Figure 2: Popular update formulas, source: Wikipedia

4 Results and comparison

Below are the results for running SGD, ADAM and L-BFGS on small dataset. As we can see that L-BFGS converse fast to local minima as compare to SGD and ADAM.

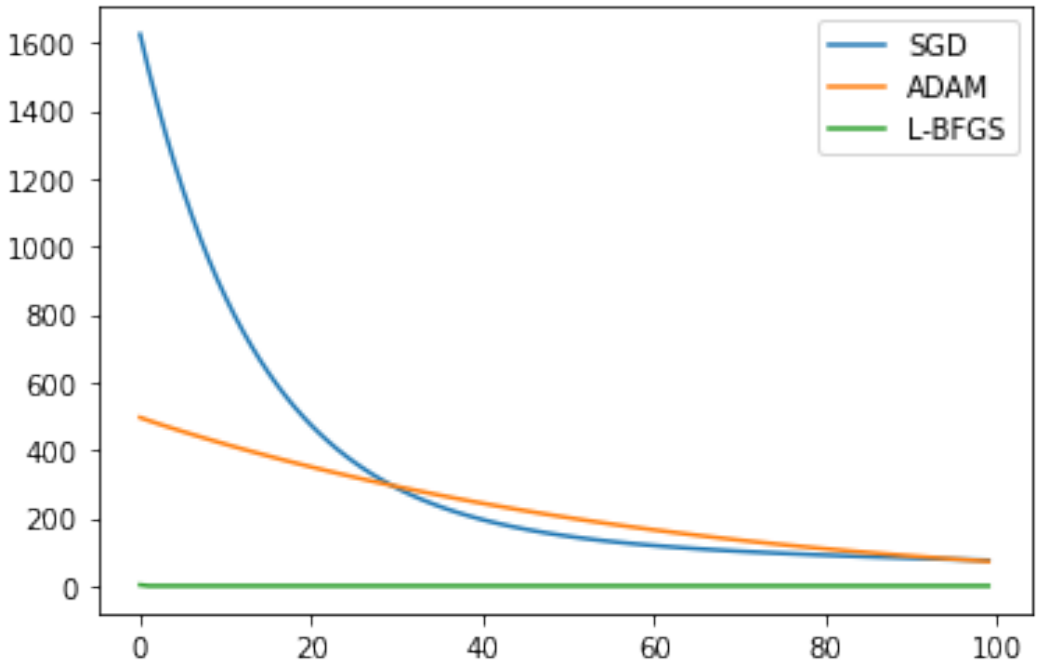

Figure 3: Result comparison on simple data set

I also run the comparison between BFGS, L-BFGS, Newton Conjugate Gradient and Original Newton Method. And original newton methods converse very fast to global minima as compare to other methods.

BFGS took 37 iteration while L-BFGS took 38 iteration duo to limited memory constraint. Also Newton Conjugate Gradient took 29 iteration while original newton only took 20 iterations.

Below is the results from a paper name Quasi-Newton Methods for Machine Learning: Forget the Past, Just Sample by A. S. Berahasa, M. Jahanib, P. Richt and M. Tak.

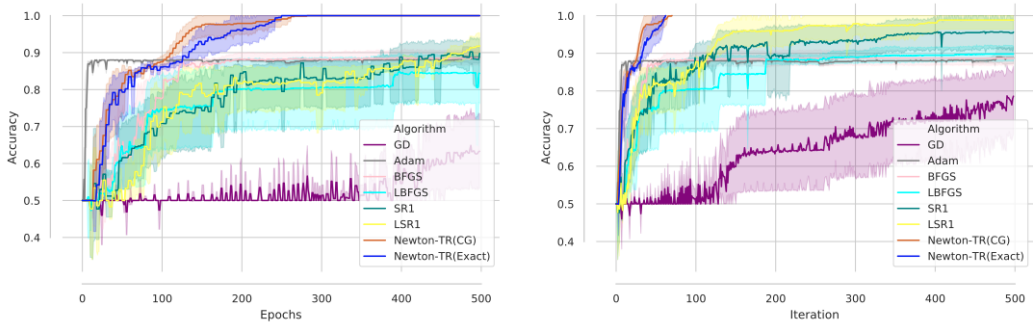

Figure 4: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, Newton-TR(CG, Exact) toy classification problem on a toy classification problem.

5 Conclusion

The findings in Figure 3 and Figure 4 show the performance (for 10 distinct beginning points) of multiple stochastic and deterministic, first- and second-order algorithms on a toy neural network classification job and dummy data. As the data show, first-order approaches converge slowly and occasionally fail to attain 100 percent accuracy. Similarly, traditional quasi-Newtonian approaches are slow and stagnant. Methods that use the genuine Hessian, on the other hand, can converge in a few iterations from any starting point. This appears to imply that second-order information is necessary for some neural network training tasks, and that the curvature information acquired by traditional quasi-Newton approaches may not be sufficient.

References

[1] Stephen Boyd and Lieven Vandenberghe. Convex Optimization. Cambridge University Express 2004.
[2] Optimization for Machine Learning Lecture Notes CS-439, Spring 2022 Bernd Gartner, ETH ¨ Martin Jaggi, EPFL
[3] Introductory Lectures on Convex Programming Volume I: Basic course Yu. Nesterov July 2, 1998
[4] John Dennis, Jorge Moré. Quasi-Newton Methods, Motivation and Theory. SIAM Review, Society for Industrial and Applied Mathematics, 1977, 19 (1), pp.46-89. ff10.1137/1019005ff. ffhal-01495720
[5] Optimization with the Quasi-Newton Method by Ronald Schoenberg September 5, 2001 Aptech Systems, Inc. Maple Valley, WA
[6] Optimization Methods for Large-Scale Machine Learning L´eon Bottou Frank E. Curtis† Jorge Nocedal‡ February 12, 2018