Notes: L-BFGS and Second-Order Optimization Methods

Nicholas Fleischhauer

October 22, 2025

1 Overview

This document explores the mathematical foundations of second-order optimization methods, with a particular focus on the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm. We compare first-order methods (gradient descent and stochastic gradient descent) with second-order methods (Newton's method and L-BFGS), examining how curvature information accelerates convergence and why certain regularization techniques are incompatible with quasi-Newton methods.

2 Key Concepts

2.1 Newton's Method vs. Gradient Descent

2.1.1 Newton's Method (Full)

Newton's method uses the **second-order Taylor approximation** to find the optimal parameter update:

$$\theta_{t+1} = \theta_t - H^{-1}\nabla f(\theta_t) \tag{1}$$

where:

- $\nabla f(\theta_t)$ is the gradient (first derivative)—the direction of steepest ascent
- H is the **Hessian matrix** (matrix of second derivatives)—encodes curvature information
- H^{-1} is the inverse Hessian matrix

Definition 1 (Hessian Matrix)**.** *For a function $f : \mathbb{R}^n \to \mathbb{R}$, the Hessian matrix $H \in \mathbb{R}^{n\times n}$ is defined as:*

$$H_{ij} = \frac{\partial^2 f}{\partial\theta_i \partial\theta_j}$$

Key insight: The Hessian tells you not just which direction to go (gradient), but also **how the gradient itself is changing** (curvature). This allows for much smarter step sizes and adaptive scaling in different directions.

2.1.2 Gradient Descent

Gradient descent uses only first-order information:

$$\theta_{t+1} = \theta_t - \alpha\nabla f(\theta_t) \tag{2}$$

where α is the learning rate (step size).

Key limitation: Only uses first-order information (gradient). It's analogous to walking down a hill blindfolded—you can feel the slope, but you don't know if the terrain is about to flatten out or get steeper.

2.2 Why GD/SGD Don't Use Curvature

No, they don't! This is the crucial difference:

- **GD/SGD**: Only compute ∇f (first derivatives)—$O(n)$ complexity per iteration
- **Newton's Method**: Computes both ∇f and H (second derivatives)—$O(n^2)$ to compute the Hessian, $O(n^3)$ to invert it

This is why Newton's method converges in far fewer iterations but becomes prohibitively expensive for high-dimensional problems (e.g., deep neural networks with millions of parameters).

3 The L-BFGS Algorithm

3.1 The Problem with Full Newton's Method

For n parameters:

- Hessian H is an $n \times n$ matrix
- For 10,000 parameters: 100 million entries!
- Computing the Hessian: $O(n^2)$ storage, $O(n^2)$ computation
- Inverting the Hessian: $O(n^3)$ operations

This becomes computationally infeasible for modern machine learning applications.

3.2 The Solution: L-BFGS

Instead of storing and computing the full Hessian, L-BFGS:

1. **Stores only the last m gradient differences and parameter differences** (typically $m = 5$ to 20)
2. Uses these vectors to **implicitly approximate** $H^{-1}\nabla f$ without ever forming H or H^{-1} explicitly

3.3 The Mathematical Foundation

L-BFGS maintains two sets of vectors from recent iterations:

$$s_k = \theta_{k+1} - \theta_k \quad \text{(change in parameters)} \tag{3}$$
$$y_k = \nabla f(\theta_{k+1}) - \nabla f(\theta_k) \quad \text{(change in gradients)} \tag{4}$$

For the last m iterations, we store the pairs: $(s_0, y_0), (s_1, y_1), \ldots, (s_{m-1}, y_{m-1})$

Storage requirement: $2m \times n$ scalars instead of n^2 (massive reduction!)

Example 2 (Storage Comparison)**.** *For $n = 10{,}000$ parameters and $m = 10$ history:*

- *Full Hessian:* $10{,}000^2 = 100{,}000{,}000$ *entries*

- *L-BFGS:* $2 \times 10 \times 10{,}000 = 200{,}000$ *entries*
- ***Reduction factor**: 500× less memory!*

3.4 Two-Loop Recursion Algorithm

This is the core computational engine of L-BFGS. It computes the search direction $d = -H^{-1}\nabla f$ using only the stored vector pairs.

Algorithm 1 L-BFGS Two-Loop Recursion

Input: Current gradient $\nabla f(\theta)$, stored pairs (s_i, y_i) for $i = 0, \ldots, m-1$
Output: Search direction d

$q \leftarrow \nabla f(\theta)$
▷ First loop: backward pass
for $i = m-1, m-2, \ldots, 0$ **do**
    $\alpha_i \leftarrow \frac{s_i^\top q}{y_i^\top s_i}$
    $q \leftarrow q - \alpha_i y_i$
end for

$r \leftarrow H_0^{-1} q$ ▷ Initial Hessian approximation (often identity)
▷ Second loop: forward pass
for $i = 0, 1, \ldots, m-1$ **do**
    $\beta \leftarrow \frac{y_i^\top r}{y_i^\top s_i}$
    $r \leftarrow r + s_i(\alpha_i - \beta)$
end for

return $d = -r$ ▷ Search direction

Complexity Analysis:

- **Space complexity**: $O(mn)$ instead of $O(n^2)$
- **Time complexity per iteration**: $O(mn)$ instead of $O(n^3)$

3.5 Why This Works: The Secant Condition

The BFGS update formula builds up an approximation $B_k \approx H$ (or its inverse) using the **secant condition**:

$$B_{k+1} s_k = y_k \tag{5}$$

This condition states: "The approximate Hessian times the change in parameters should equal the change in gradients."

This is derived from the mean value theorem applied to the gradient:

$$\nabla f(\theta_{k+1}) - \nabla f(\theta_k) \approx H(\theta_k)(\theta_{k+1} - \theta_k) \tag{6}$$

Rearranging:

$$y_k \approx H \cdot s_k \tag{7}$$

The secant condition ensures that our Hessian approximation is consistent with the observed changes in the gradient along the optimization trajectory.

4 Convergence Analysis

4.1 Geometric Intuition

Consider optimizing a function with an elongated valley (high condition number, such as a poorly conditioned quadratic):

Gradient Descent:

- Sees steep gradient perpendicular to the valley
- Takes a step perpendicular to the valley, overshoots
- Zigzags back and forth across the valley (thousands of iterations)

L-BFGS:

- Curvature information reveals: "steep in this direction, flat in that direction"
- Takes a **preconditioned step**—small in steep directions, large in flat directions
- Directly moves down the valley axis (tens of iterations)

4.2 Convergence Rates

The convergence rate depends critically on the **condition number** $\kappa = \lambda_{\max}/\lambda_{\min}$ of the Hessian, where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues, respectively.

Theorem 3 (Convergence Rates)**.** *For strongly convex functions:*

- ***Gradient Descent**: Linear convergence with rate depending on κ*

$$\|\theta_t - \theta^*\| \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^t \|\theta_0 - \theta^*\|$$

  Number of iterations: $O(\kappa \log(1/\epsilon))$ to achieve ϵ accuracy

- ***Newton's Method / L-BFGS**: Superlinear to quadratic convergence*

$$\|\theta_{t+1} - \theta^*\| \leq C\|\theta_t - \theta^*\|^2$$

  Number of iterations: $O(\log\log(1/\epsilon))$ to achieve ϵ accuracy

For poorly conditioned problems (large κ), the difference is dramatic:

- $\kappa = 1000$: GD needs $\sim$1000$\times$ more iterations
- L-BFGS convergence is nearly independent of κ

5 Why L-BFGS Cannot Handle L1 Regularization

5.1 The Problem with L1

Consider **L1 regularization** (Lasso):

$$f(\theta) = \text{loss}(\theta) + \lambda\|\theta\|_1 = \text{loss}(\theta) + \lambda \sum_{i=1}^{n} |\theta_i| \tag{8}$$

The critical issue: **The L1 norm is not differentiable at zero**.

$$\frac{\partial|\theta|}{\partial\theta} = \begin{cases} +1 & \text{if } \theta > 0 \\ -1 & \text{if } \theta < 0 \\ \text{undefined} & \text{if } \theta = 0 \end{cases} \tag{9}$$

5.2 Why L-BFGS Specifically Fails

1. **L-BFGS assumes smoothness**: The entire BFGS approximation relies on the objective function being twice continuously differentiable (C^2). The secant condition $B_k s_k = y_k$ assumes gradients change smoothly along the optimization path.

2. **Kinks break the Hessian approximation**: At $\theta = 0$, the gradient has a discontinuous jump (from -1 to $+1$, or vice versa). The Hessian approximation becomes invalid at these non-smooth points.

3. **Cannot identify exact zeros**: L1 regularization drives coefficients to **exactly zero** (inducing sparsity), which is its key feature. L-BFGS will approach zero asymptotically but won't hit it exactly because it relies on smooth curvature information.

4. **Subgradients are insufficient**: While we could use subgradients, the secant condition $B_k s_k = y_k$ loses its theoretical justification when y_k contains arbitrary subgradient selections rather than true gradients.

5.3 What Works Instead: Proximal Methods

For non-smooth regularizers like L1, we use **proximal gradient methods** (implemented in solvers like SAGA):

$$\theta_{t+1} = \text{prox}_{\lambda\|\cdot\|_1}\left(\theta_t - \alpha\nabla\text{loss}(\theta_t)\right) \tag{10}$$

where the proximal operator for L1 regularization is **soft thresholding**:

$$\text{prox}_{\lambda|\theta|}(\theta) = \text{sign}(\theta)\max(|\theta| - \lambda, 0) = \begin{cases} \theta - \lambda & \text{if } \theta > \lambda \\ 0 & \text{if } |\theta| \leq \lambda \\ \theta + \lambda & \text{if } \theta < -\lambda \end{cases} \tag{11}$$

This approach explicitly separates:

- The **smooth** differentiable loss term (handled by gradient descent)
- The **non-smooth** L1 regularizer (handled by the proximal operator)

Remark 4 (L2 Regularization)**.** *In contrast, L2 regularization (Ridge) is perfectly compatible with L-BFGS because:*

$$f(\theta) = loss(\theta) + \lambda\|\theta\|_2^2$$

is smooth and twice differentiable everywhere. The gradient is:

$$\nabla f(\theta) = \nabla loss(\theta) + 2\lambda\theta$$

which is continuous and differentiable.

6 Summary: The Core Advantages of L-BFGS

The computational efficiency and rapid convergence of L-BFGS stem from:

1. **Using curvature information**: Exploits second-order (Hessian) information that first-order methods (GD/SGD) completely ignore
2. **Implicit Hessian representation**: Stores the inverse Hessian implicitly using only m vector pairs, avoiding $O(n^2)$ storage and $O(n^3)$ inversion
3. **Two-loop recursion**: Computes the search direction in $O(mn)$ time through clever recursive application of stored vector pairs
4. **Superlinear convergence**: Steps naturally adapt to the loss surface geometry, achieving convergence rates nearly independent of the condition number
5. **No learning rate tuning**: Unlike gradient descent, L-BFGS uses line search to automatically determine step sizes

Critical limitation: Requires smooth, twice-differentiable objective functions, which is why non-smooth regularizers (L1, elastic net) break the algorithm and require specialized proximal methods.

7 Notes and Observations

7.1 When to Use L-BFGS

L-BFGS is ideal for:

- Medium-scale problems (thousands to millions of parameters)
- Smooth, well-conditioned objective functions
- Problems where gradient computation is expensive (fewer iterations matter)
- L2-regularized models (Ridge regression, logistic regression)

7.2 When NOT to Use L-BFGS

Avoid L-BFGS for:

- Very large-scale problems (deep learning with billions of parameters)—storage of m vector pairs becomes prohibitive
- Non-smooth objectives (L1 regularization, hinge loss, etc.)
- Stochastic objectives where gradients are noisy (use SGD variants instead)
- Problems requiring online learning or streaming data

7.3 Practical Considerations

- **History size** m: Typical values are 5–20. Larger m improves approximation but increases computation and memory
- **Initial Hessian** H_0: Often set to identity or scaled identity based on recent s and y pairs
- **Line search**: L-BFGS requires a line search (e.g., Wolfe conditions) to ensure sufficient decrease and curvature conditions

8 References

- Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization* (2nd ed.). Springer.
- Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3), 503–528.
- Parikh, N., & Boyd, S. (2014). Proximal algorithms. *Foundations and Trends in Optimization*, 1(3), 127–239.

