Lecture 23: Limited-Memory BFGS (L-BFGS)

Yudong Chen

1 Basic ideas

Newton and quasi-Newton methods enjoy fast convergence (small number of iterations), but for large-scale problems each iteration may be too costly.

For example, recall the quasi-Newton method

\[ x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k) \]

with BFGS update:

\[ H_k = V_{k-1}^T V_{k-1} + \rho_k^{-1} s_{k-1} s_{k-1}^T, \]

where

\[ \rho_k = \frac{1}{s_k^T y_k}, \quad V_k = I - \rho_k y_k s_k^T, \]

\[ s_k = x_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k), \]

and the stepsize \( \alpha_k \) satisfies WWC. The matrices \( B_k \) and \( H_k \) constructed by BFGS are often dense, even when the true Hessian is sparse. In general, BFGS requires \( \Theta(d^2) \) computation per iteration and \( \Theta(d^2) \) memory. For large \( d \), \( \Theta(d^2) \) may be too much.

**Idea of L-BFGS:** instead of storing the full matrix \( H_k \) (approximation of \( \nabla^2 f(x_k)^{-1} \)), construct and represent \( H_k \) implicitly using a small number of vectors \( \{s_i, y_i\} \) for the last few iterations.

**Intuition:** we do not expect the current Hessian to depend too much on “old” vectors \( s_i, y_i \) (old iterates \( x_i \) and their gradients.)

**Tradeoff:** we reduce memory and computation to \( O(d) \), but we may lose local superlinear convergence—we can only guarantee linear convergence in general.

2 L-BFGS

Recall and expand the BFGS update:

\[
H_k = V_{k-1}^T V_{k-1} + \rho_k^{-1} s_{k-1} s_{k-1}^T \\
= V_{k-1}^T V_{k-2} H_{k-2} V_{k-2} V_{k-1} + \rho_k^{-1} s_{k-2} s_{k-2}^T V_{k-1} + \rho_k^{-1} s_{k-1} s_{k-1}^T \\
= \left( V_{k-1}^T \cdots V_{k-m}^T \right) H_{k-m} \left( V_{k-m} V_{k-m+1} \cdots V_{k-1} \right) \\
+ \rho_k^{-1} \left( V_{k-1}^T \cdots V_{k-m+1}^T \right) s_{k-m} s_{k-m}^T \left( V_{k-m+1} \cdots V_{k-1} \right) \\
+ \rho_k^{-1} \left( V_{k-1}^T \cdots V_{k-m+2}^T \right) s_{k-m+1} s_{k-m+1}^T \left( V_{k-m+2} \cdots V_{k-1} \right) \\
+ \cdots \\
+ \rho_k^{-1} V_{k-m} V_{k-m+1} V_{k-m+2} \cdots V_{k-1} \\
+ \rho_k^{-1} s_{k-1} s_{k-1}^T.
\]
In L-BFGS, we replace $H_{k-m}$ (a dense $d \times d$ matrix) with some sparse matrix $H_0^0$, e.g., a diagonal matrix. Thus, $H_k$ can be constructed using the most recent $m \ll d$ pairs $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-m}^{k-1}$. That is,

$$L-BFGS: \quad H_k = \left( V_{k-1}^T V_{k-2}^T \cdots V_{k-m}^T \right) H_0^0 \left( V_{k-m} V_{k-m+1} \cdots V_{k-1} \right) + \rho_{k-m} \left( V_{k-1}^T \cdots V_{k-m+1}^T \right) \mathbf{s}_{k-m} \mathbf{y}_{k-m} \left( V_{k-m+1} \cdots V_{k-1} \right) + \rho_{k-m+1} \left( V_{k-1}^T \cdots V_{k-m+2}^T \right) \mathbf{s}_{k-m+1} \mathbf{y}_{k-m+1} \left( V_{k-m+2} \cdots V_{k-1} \right) + \cdots + \rho_{k-1} \mathbf{s}_{k-1} \mathbf{y}_{k-1}.$$

In fact, we only need the $d$-dimensional vector $H_k \nabla f(x_k)$ to update $x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k)$. Therefore, we do not even need to compute or store the matrix $H_k$ explicitly. Instead, we only store the vectors $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-m'}$ from which $H_k \nabla f(x_k)$ can be computed using only vector-vector multiplications, thanks to tricks like $(a a^T + b b^T)g = a(a^T g) + b(b^T g)$.

This leads to a two-loop recursion implementation for computing $H_k \nabla f(x_k)$, stated in Algorithm 1.

**Algorithm 1** L-BFGS two-loop recursion

```plaintext
set $q = \nabla f(x_k)$ want to compute $H_k \cdot \nabla f(x_k)$
for $i = k-1, k-2, \ldots, k = m$ do:
    $\alpha_i \leftarrow \rho_i \mathbf{s}_i^T q$
    $q \leftarrow q - \alpha_i \mathbf{y}_i$ // RHS = $q - \rho_i \mathbf{s}_i^T \mathbf{y}_i = \left( I - \rho_i \mathbf{s}_i \mathbf{s}_i^T \right) q$
    $r = H_k^0 q$
for $i = k - m$ to $k - 1$:
    $\beta \leftarrow \rho_i \mathbf{y}_i^T r$
    $r \leftarrow r + \mathbf{s}_i (\alpha_i - \beta)$ // RHS = $r + \rho_i \mathbf{a}_i - \rho_i \mathbf{y}_i^T r s_i = \left( I - \rho_i \mathbf{s}_i \mathbf{s}_i^T \right) r + \rho_i \mathbf{a}_i$
return $r$ // which equals $H_k \nabla f(x_k)$
```

(Exercise) The total number of multiplications is at most $4md + \text{nnz}(H_0^0) = O(md)$.

In practice:

- We often take $m$ to be a small constant independent of $d$, e.g., $3 \leq m \leq 20$.
- A popular choice for $H_0^0$ is $H_0^0 = \gamma_k I$, where $\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$. This choice appears to be quite effective in practice. (Optional) $\frac{1}{\gamma_k}$ is an approximation of $\frac{s_{k}^T \nabla^2 f(x_k) z_k}{\|z_k\|^2}$, which is the size of the true Hessian along the direction $z_k \approx \left( \nabla^2 f(x_k) \right)^{1/2} s_k$; see Section 6.1 in Nocedal-Wright.

The complete L-BFGS algorithm is given in Algorithm 2. As discussed in Lecture 21, it is important that $\alpha_k$ satisfies both the sufficient decrease and curvature conditions in Wolfe.
Algorithm 2 L-BFGS

**input:** $x_0 \in \mathbb{R}^d$ (initial point), $m > 0$ (memory budget), $\epsilon > 0$ (convergence criterion)

$k \leftarrow 0$

**repeat:**

- Choose $H_0^k$
- $p_k \leftarrow -H_k \nabla f(x_k)$, where $H_k \nabla f(x_k)$ is computed using Algorithm 1
- $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where $\alpha_k$ satisfies Wolfe Conditions
- **if** $k > m$:
  - discard $\{s_{k-m}, y_{k-m}\}$ from storage
- Compute and store $s_k \leftarrow x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
- $k \leftarrow k + 1$

**until** $\|\nabla f(x_k)\| \leq \epsilon$

Some numerical results taken from Nocedal-Wright:

Table 7.1 presents results illustrating the behavior of Algorithm 7.5 for various levels of memory $m$. It gives the number of function and gradient evaluations (nfg) and the total CPU time. The test problems are taken from the CUTE collection [35], the number of variables is indicated by $n$, and the termination criterion $\|\nabla f_k\| \leq 10^{-5}$ is used. The table shows that the algorithm tends to be less robust when $m$ is small. As the amount of storage increases, the number of function evaluations tends to decrease; but since the cost of each iteration increases with the amount of storage, the best CPU time is often obtained for small values of $m$. Clearly, the optimal choice of $m$ is problem dependent.

<table>
<thead>
<tr>
<th>Problem</th>
<th>$n$</th>
<th>L-BFGS $m = 3$</th>
<th>L-BFGS $m = 5$</th>
<th>L-BFGS $m = 17$</th>
<th>L-BFGS $m = 29$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nfg</td>
<td>time</td>
<td>nfg</td>
<td>time</td>
<td>nfg</td>
</tr>
<tr>
<td>DIXMAANL</td>
<td>1500</td>
<td>146 16.5</td>
<td>134 17.4</td>
<td>120 28.2</td>
<td>125 44.4</td>
</tr>
<tr>
<td>EIGENALS</td>
<td>110</td>
<td>821 21.5</td>
<td>569 15.7</td>
<td>363 16.2</td>
<td>168 12.5</td>
</tr>
<tr>
<td>FREUROTH</td>
<td>1000</td>
<td>&gt;999 —</td>
<td>&gt;999 —</td>
<td>69 8.1</td>
<td>38 6.3</td>
</tr>
<tr>
<td>TRIDIA</td>
<td>1000</td>
<td>876 46.6</td>
<td>611 41.4</td>
<td>531 84.6</td>
<td>462 127.1</td>
</tr>
</tbody>
</table>

3 Relationship with nonlinear conjugate gradient methods

In Lecture 13 we mentioned several ways of generalizing CG to non-quadratic functions (a.k.a. non-linear CG), including Dai-Yuan, Fletcher-Rieves and Polak-Ribiere. The last one has a variant
called Hestenes-Stiefel, which uses the search direction

\[ p_{k+1} = -\nabla f(x_{k+1}) + \frac{\nabla f(x_{k+1})^\top y_k}{y_k^\top p_k} p_k = -\left( I - \frac{s_k y_k^\top}{y_k^\top s_k} \right) \nabla f(x_{k+1}), \]  

(2)

where we recall that \( y_k = \nabla f(x_{k+1}) - \nabla f(x_k) \) and \( s_k = x_{k+1} - x_k \).

The matrix \( \hat{H}_{k+1} \) is neither symmetric nor p.d. If we try to symmetrize \( \hat{H}_{k+1} \) by taking \( \hat{H}_{k+1}^\top \hat{H}_{k+1} \), we end up with a matrix that does not satisfy the secant equation and is singular.

A symmetric p.d. matrix that satisfies the secant equation is

\[ H_{k+1} = \hat{H}_{k+1} \hat{H}_{k+1}^\top + \frac{s_k s_k^\top}{y_k^\top s_k} \]

\[ = \left( I - \frac{s_k y_k^\top}{y_k^\top s_k} \right) \left( I - \frac{y_k s_k^\top}{y_k^\top s_k} \right) + \frac{s_k s_k^\top}{y_k^\top s_k} \]

\[ = \text{BFGS update (1) applied to } H_k = I \]

Therefore, computing \( H_{k+1} \) as above for the search direction \( p_{k+1} = -H_{k+1} \nabla f(x_{k+1}) \) can be viewed as “memoryless” BFGS, i.e., L-BFGS with \( m = 1 \) and \( H_0^0 = I \).

Suppose we combine memoryless BFGS and exact line search:

\[ \alpha_k = \arg\min_{\alpha \in \mathbb{R}} f(x_k + \alpha p_k). \]

For all \( k \), the stepsize \( \alpha_k \) satisfies

\[ 0 = \langle \nabla f(x_k + \alpha_k p_k), p_k \rangle = \langle \nabla f(x_{k+1}), \alpha_k^{-1} s_k \rangle, \]

hence \( s_k^\top \nabla f(x_{k+1}) = 0 \). It follows that

\[ p_{k+1} = -H_{k+1} \nabla f(x_{k+1}) \]

\[ = -\left[ \left( I - \frac{s_k y_k^\top}{y_k^\top s_k} \right) \left( I - \frac{y_k s_k^\top}{y_k^\top s_k} \right) + \frac{s_k s_k^\top}{y_k^\top s_k} \right] \nabla f(x_{k+1}) \]

\[ = -\nabla f(x_{k+1}) + \frac{y_k^\top \nabla f(x_{k+1}) s_k}{y_k^\top s_k} s_k \]

\[ = -\nabla f(x_{k+1}) + \frac{y_k^\top \nabla f(x_{k+1})}{y_k^\top p_k} p_k, \]

\[ s_k = \alpha_k p_k \]

which is the same as Hestenes-Stiefel CG update (2).