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:

$$H_k = V_{k-1}^\top H_{k-1} V_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^\top, \tag{1}$$

where

$$\rho_k = \frac{1}{s_k^\top y_k}, \qquad V_k = I - \rho_k y_k s_k^\top,$$
$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla f(x_{k+1}) - \nabla f(x_k),$$

and the stepsize α_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:

$$\begin{aligned}
\text{BFGS:} \quad H_k =& V_{k-1}^\top H_{k-1} V_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^\top \\
=& V_{k-1}^\top V_{k-2}^\top H_{k-2} V_{k-2} V_{k-1} + \rho_{k-2} V_{k-2} s_{k-2} s_{k-2}^\top V_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^\top \\
=& \left(V_{k-1}^\top V_{k-2}^\top \cdots V_{k-m}^\top \right) H_{k-m} \left(V_{k-m} V_{k-m+1} \cdots V_{k-1} \right) \\
&+ \rho_{k-m} \left(V_{k-1}^\top \cdots V_{k-m+1}^\top \right) s_{k-m} s_{k-m}^\top \left(V_{k-m+1} \cdots V_{k-1} \right) \\
&+ \rho_{k-m+1} \left(V_{k-1}^\top \cdots V_{k-m+2}^\top \right) s_{k-m+1} s_{k-m+1}^\top \left(V_{k-m+2} \cdots V_{k-1} \right) \\
&+ \cdots \\
&+ \rho_{k-2} V_{k-1}^\top s_{k-2} s_{k-2}^\top V_{k-1} \\
&+ \rho_{k-1} s_{k-1} s_{k-1}^\top.
\end{aligned}$$

In L-BFGS, we replace H_{k-m} (a dense $d \times d$ matrix) with some sparse matrix H_k^0, e.g., a diagonal matrix. Thus, H_k can be constructed using the most recent $m \ll d$ pairs $\{s_i, y_i\}_{i=k-m}^{k-1}$. That is,

$$\begin{aligned}
\text{L-BFGS:} \qquad H_k = & \left(V_{k-1}^\top V_{k-2}^\top \cdots V_{k-m}^\top\right) H_k^0 \left(V_{k-m} V_{k-m+1} \cdots V_{k-1}\right) \\
& + \rho_{k-m} \left(V_{k-1}^\top \cdots V_{k-m+1}^\top\right) s_{k-m} s_{k-m}^\top \left(V_{k-m+1} \cdots V_{k-1}\right) \\
& + \rho_{k-m+1} \left(V_{k-1}^\top \cdots V_{k-m+2}^\top\right) s_{k-m+1} s_{k-m+1}^\top \left(V_{k-m+2} \cdots V_{k-1}\right) \\
& + \cdots \\
& + \rho_{k-1} s_{k-1} s_{k-1}^\top.
\end{aligned}$$

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 $\{s_i, y_i\}_{i=k-m}^{k-1}$, from which $H_k \nabla f(x_k)$ can be computed using only vector-vector multiplications, thanks to tricks like $(aa^\top + bb^\top)g = a(a^\top g) + b(b^\top g)$.

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

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**:
  $\alpha_i \leftarrow \rho_i s_i^\top q$
  $q \leftarrow q - \alpha_i y_i$ // RHS$= q - \rho_i s_i^\top q y_i = \underbrace{\left(I - \rho_i y_i s_i^\top\right)}_{V_i} q$
$r = H_k^0 q$
for $i = k - m$ to $k - 1$:
  $\beta \leftarrow \rho_i y_i^\top r$
  $r \leftarrow r + s_i(\alpha_i - \beta)$ // RHS $= r + \rho_i \alpha_i - \rho_i y_i^\top r s_i = \underbrace{\left(I - \rho_i s_i y_i^\top\right)}_{V_i^\top} r + \rho_i \alpha_i$
return r // which equals $H_k \nabla f(x_k)$

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

- We often take m to be a small constant independent of d, e.g., $3 \le m \le 20$.
- A popular choice for H_k^0 is $H_k^0 = \gamma_k I$, where $\gamma_k = \frac{s_{k-1}^\top y_{k-1}}{y_{k-1}^\top y_{k-1}}$. This choice appears to be quite effective in practice. (Optional) $\frac{1}{\gamma_k}$ is an approximation of $\frac{z_k^\top \nabla^2 f(x_k) z_k}{\|z_k\|^2}$, 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 α_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_k^0
- $p_k \leftarrow -H_k \nabla f(x_k)$, where $H_k \nabla f(x_k)$ is computed using Algorithm 1
- $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where α_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 7.1 Performance of Algorithm 7.5.

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

3 Relationship with nonlinear conjugate gradient methods

In Lecture 13 we mentioned several ways of generalizing CG to non-quadratic functions (a.k.a. nonlienar CG), including Dai-Yuan, Fletcher-Rieves and Polak-Ribiere. The last one has a variant

called Hestenes-Stiefel, which uses the search direction

$$p_{k+1} = -\nabla f(x_{k+1}) + \frac{\nabla f(x_{k+1})^\top y_k}{y_k^\top p_k} p_k = -\underbrace{\left(I - \frac{s_k y_k^\top}{y_k^\top s_k}\right)}_{=:\hat{H}_{k+1}} \nabla f(x_{k+1}), \tag{2}$$

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

$$\begin{aligned} H_{k+1} &= \hat{H}_{k+1}\hat{H}_{k+1}^\top + \frac{s_k s_k^\top}{y_k^\top s_k} \\ &= \left(I - \frac{s_k y_k^\top}{y_k^\top s_k}\right) I \left(I - \frac{y_k s_k^\top}{y_k^\top s_k}\right) + \frac{s_k s_k^\top}{y_k^\top s_k} \\ &= \text{BFGS update (1) applied to } H_k = I \end{aligned}$$

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 $H_k^0 = I$.

Suppose we combine memoryless BFGS and exact line search:

$$\alpha_k = \operatorname*{argmin}_{\alpha \in \mathbb{R}} f(x_k + \alpha p_k).$$

For all k , the stepsize α_k satisfies

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

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

$$\begin{aligned} p_{k+1} &= -H_{k+1}\nabla f(x_{k+1}) \\ &= -\left[\left(I - \frac{s_k y_k^\top}{y_k^\top s_k}\right)\left(I - \frac{y_k s_k^\top}{y_k^\top s_k}\right) + \frac{s_k s_k^\top}{y_k^\top s_k}\right] \nabla f(x_{k+1}) \\ &= -\nabla f(x_{k+1}) + \frac{y_k^\top \nabla f(x_{k+1})}{y_k^\top s_k} s_k && s_k^\top \nabla f(x_{k+1}) = 0 \\ &= -\nabla f(x_{k+1}) + \frac{y_k^\top \nabla f(x_{k+1})}{y_k^\top p_k} p_k, && s_k = \alpha_k p_k \end{aligned}$$

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