مجله مدل‌سازی پیشرفته ریاضی
سال ۱۴۰۴، دوره ۱۵، شماره ۱، ص ۷۸–۹۱

شاپا چاپی: ۸۰۸۸-۲۲۵۱
DOI: 10.22055/jamm.2025.48314.2330

ارائه‌ی یک روش بهینه‌سازی شبه‌نیوتن با حافظه‌ی محدود و قابلیت اجرا به‌صورت موازی

اشکان صادقی لطف آبادی [۱] سید کمال‌الدین غیاثی شیرازی [۱,۱]*

[۱] گروه مهندسی کامپیوتر، دانشکده مهندسی، دانشگاه فردوسی مشهد، مشهد، ایران

دبیر مسئول: محمد هادی فراهی

تاریخ دریافت: ۱۴۰۳/۰۸/۲۷ تاریخ پذیرش: ۱۴۰۴/۰۳/۰۷

چکیده: روش نیوتن به‌دلیل دقت بالاتر نسبت به روش‌های مرتبه اول در تقریب تابع هدف، گزینه‌ای مطلوب محسوب می‌شود؛ با این حال، نیاز به محاسبه صریح ماتریس هسی دارد. در مقابل، روش‌های شبه‌نیوتن به‌دلیل بی‌نیازی از محاسبه صریح ماتریس هسی همواره مورد توجه پژوهشگران بوده‌اند. هدف ما در اینجا ارائه یک روش شبه‌نیوتن است به‌طوری‌که بتواند در عین تقریب هسی به‌صورت مناسب، سرعت بالایی هم داشته باشد. روش ما از دید عملگری به ماتریس هسی نگاه می‌کند. طبق دیدگاه عملگری، ماتریس مانند یک عملگر خطی است که یک بردار ورودی دریافت و یک بردار در خروجی به‌دست می‌دهد. هسی نیز طبق این دیدگاه یک بردار گرادیان را دریافت و یک بردار در خروجی به‌دست می‌دهد. در روش ما هسی نه به‌صورت صریح، بلکه به‌صورت تقریب رتبه پایین و به‌وسیله‌ی یک سری بردار و عدد نمایش داده می‌شود. روش ما قابلیت انجام محاسبات به‌صورت موازی را دارد و به همین دلیل سرعت اجرای بالایی دارد. افزون‌بر این، روش ما توانایی کار با حافظه محدود را دارد. نتایج به‌دست آمده نیز نشان می‌دهد که روش ارائه شده، نسبت به روش شبه‌نیوتن L–BFGS سرعت بیشتری دارد. همچنین در مسائلی که تعداد زیادی متغیر دارند، روش ما نسبت به BFGS از کارایی محاسباتی بیشتری برخوردار است.

واژه‌های کلیدی: شبه‌نیوتن، L–BFGS ، دیدگاه عملگری، تقریب رتبه پایین.

رده‌بندی ریاضی: 90C53; 65K05; 49M37

[۱]* نویسنده مسئول مقاله
(K. Ghiasi-Shirazi) k.ghiasi@um.ac.ir, (A. Sadeghi-Lotfabadi) sadeghia@mail.um.ac.ir

۱ مقدمه

در بهینه‌سازی نامقید هدف یافتن یک پارامتر بهینه w^* برای یک تابع مانند $f(w)$ است:

$$w^* = \arg\min_{w \in R^d} f(w). \quad (۱.۱)$$

در روش‌های جستجوی خطی، برای به‌دست آوردن پارامتر بهینه از فرمول

$$w_{k+۱} = w_k + \alpha_k p_k, \quad (۲.۱)$$

استفاده می‌شود. در این فرمول α_k طول گام است و p_k جهت حرکت را نشان می‌دهد و اغلب به‌صورت

$$p_k = -B_k^{-۱} \nabla f_k, \quad (۳.۱)$$

تعریف می‌شود که در این عبارت B_k یک ماتریس متقارن و ناتکین است [۱].
به‌دست آوردن ماتریس B_k و نحوه ذخیره‌سازی آن یکی از چالش‌های روش‌های بهینه‌سازی است. معکوس ماتریس B_k روی گرادیان تابع (∇f_k) اعمال می‌شود و خروجی، جهت بهینه‌سازی و نحوه به‌روزرسانی پارامتر w را تعیین می‌کند. نحوه انتخاب B_k منجر به روش‌های بهینه‌سازی متفاوتی می‌شود. روش نیوتن B_k را $\nabla^۲ f_k$ در نظر می‌گیرد که همان ماتریس هسی [۲] است. اما به‌دست آوردن هسی و مشتق مرتبه دوم از نظر محاسباتی به‌خصوص در ابعاد بالا بسیار پرهزینه است. روش تندترین کاهش، B_k را یک ماتریس همانی I [۳] در نظر می‌گیرد. اما این روش تخمین دقیقی از منحنی بهینه‌سازی به‌دست نمی‌دهد. در روش‌های شبه‌نیوتن، B_k تقریبی از هسی است که در هر تکرار [۴] به‌روزرسانی می‌شود. این روش‌ها در عین حال که هسی را تقریب می‌زنند، نیازی به مشتق مرتبه دوم ندارند و فقط با در نظرگیری تغییرات گرادیان، تقریب خوبی از تابع هدف به‌دست می‌آورند.
یکی از روش‌های شبه‌نیوتن معروف، BFGS [۲–۵] است. نسخه حافظه محدود این الگوریتم با نام L–BFGS [۶، ۷] نیز ارائه شده است. روش L–BFGS از نظر محاسباتی بسیار بهینه عمل می‌کند و با نگهداری تعداد محدودی (m) از تغییرات گرادیان در لحظات قبلی می‌تواند مقدار $H_k \nabla f_k$ را به‌دست آورد که در این فرمول $H_k = B_k^{-۱}$. این مزیت بزرگ باعث شده است که کارهای زیادی در حوزه‌های مختلف به‌خصوص یادگیری ماشین و گرادیان تصادفی [۸، ۹] بر روی این الگوریتم انجام شود.
اهمیت L–BFGS باعث شد تا ما روی این الگوریتم کار کنیم. بخشی از روش L–BFGS شامل محاسبات سریالی است. محاسبه سریالی یعنی ورودی هر مرحله به خروجی مرحله قبل وابسته است و این امر باعث می‌شود تا نتوان محاسبات را به‌صورت موازی انجام داد. در L–BFGS ، این محاسبات سریالی به طول حافظه m بستگی دارد. نحوه محاسبات سریالی و وابستگی این محاسبات به m در دو حلقه for در الگوریتم ۱ قابل مشاهده است.
هدف ما حذف محاسبات سریالی از L–BFGS و جایگزینی آن‌ها با محاسباتی است که امکان موازی‌سازی داشته باشند. ما ماتریس H_k را از دید یک عملگر بررسی می‌کنیم. عملگری که در هر لحظه گرادیان ورودی را دریافت و یک گرادیان خروجی به ما برمی‌گرداند. ذخیره ماتریس H_k و کار کردن با آن هزینه زیادی دارد. ما به‌جای اینکه مستقیما از ماتریس H_k استفاده کنیم، یک تقریب رتبه پایین از آن به‌دست می‌آوریم و محاسبات و ذخیره‌سازی مقادیر را بر اساس این تقریب انجام می‌دهیم. روش ما نیز مانند L–BFGS با حافظه محدود کار می‌کند و نیاز نیست تمام مقادیر را از گام ابتدایی تا لحظه جاری نگه دارد. به‌طور کلی هدف ما به‌دست آوردن یک نمایش مانند فرمول زیر برای ماتریس H_k است:

$$H_k = \sum_{i=۱}^{r} \sigma_i u_i v_i^T. \quad (۴.۱)$$

در این معادله u_i و v_i بردارهایی d بعدی و σ_i یک اسکالر است. r هم تعداد بردارها و مقادیر لازم جهت تقریب رتبه پایین H_k است. در روش ما بردارهای u_i و v_i به گونه‌ای انتخاب می‌شوند که تقارن ماتریس H_k همواره حفظ شود. حافظه موردنیاز ما از مرتبه $O(rd)$ است. در نتیجه می‌توان گفت که میزان حافظه مصرفی الگوریتم ما نسبت به d رشد خطی دارد.

---
[۲] Hessian Matrix
[۳] Identity Matrix
[۴] Iteration

تقریب H_k به‌صورت رتبه پایین باعث می‌شود تا بتوانیم فقط با ضرب ماتریس در بردار و ماتریس در ماتریس مقدار $\nabla f_k H_k$ را محاسبه کنیم و در نتیجه الگوریتم ما بر خلاف L–BFGS ، محاسبات سریالی ندارد. در واقع ما یک مدل حافظه محدود ارائه می‌کنیم که از L–BFGS سریع‌تر و از نظر حافظه موردنیاز، هم مرتبه با L–BFGS است. آزمایش‌ها نشان می‌دهد که در برخی مسائل نظیر مساله درجه دو، روش ما تا حدود ۱۰ برابر سریع‌تر از L–BFGS عمل می‌کند. همچنین روش ما در مقایسه با BFGS در مسائل با ابعاد بالا، سریع‌تر عمل می‌کند.

ساختار مقاله به شرح زیر است. در بخش ۲ به معرفی BFGS و L–BFGS و کارهای پیشین می‌پردازیم. پس از آن در بخش ۳ روش ارائه شده را توضیح می‌دهیم. سپس در بخش ۴ آزمایش‌های انجام شده و نتایج به‌دست آمده را بررسی می‌کنیم. سرانجام در بخش ۵ به جمع‌بندی موضوع و کارهای آتی می‌پردازیم.

۲ BFGS ، L–BFGS و کارهای پیشین

روش‌های شبه‌نیوتن به‌جای محاسبه دقیق ماتریس هسی، به‌دنبال تقریبی از آن هستند. این تقریب‌ها اغلب با کمک اختلاف گرادیان‌ها به‌دست می‌آید. B یک تقریب از هسی است که در هر تکرار به‌وسیله‌ی یک فرمول رتبه پایین به‌روز می‌شود. یکی از معروف‌ترین روش‌های شبه‌نیوتن، BFGS است. BFGS به‌جای محاسبه دقیق هسی، از تقریب آن استفاده می‌کند که در هر تکرار با فرمول زیر به‌روز می‌شود:

$$B_k = B_{k-1} - \frac{B_{k-1}s_{k-1}s_{k-1}^T B_{k-1}}{s_{k-1}^T B_{k-1} s_{k-1}} + \frac{y_{k-1}y_{k-1}^T}{y_{k-1}^T s_{k-1}} \tag{۱.۲}$$

در این فرمول s_{k-1} و y_{k-1} معادل عبارات زیر هستند:

$$s_{k-1} = w_k - w_{k-1}, \tag{۲.۲}$$

$$y_{k-1} = \nabla f_k - \nabla f_{k-1}. \tag{۳.۲}$$

معادله (۲.۲) نشان‌دهنده‌ی اختلاف پارامترها در دو لحظه متوالی است. معادله (۳.۲) اختلاف گرادیان‌های دو لحظه متوالی را نشان می‌دهد. اگر H_{k-1} را معکوس B_{k-1} تعریف کنیم، آن‌گاه می‌توانیم با کمک فرمول شرمن–موریسون–وودبری (برای مثال، به کتاب [۱]، پیوست A، صفحات ۶۱۲ و ۶۱۳ بنگرید.)، فرمول (۱.۲) را به‌صورت

$$H_k = \Lambda_{k-1}^T H_{k-1} \Lambda_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^T. \tag{۴.۲}$$

بازنویسی کنیم که در این فرمول ρ_{k-1} و Λ_{k-1} برابر عبارات زیر هستند:

$$\rho_{k-1} = \frac{1}{y_{k-1}^T s_{k-1}}, \tag{۵.۲}$$

$$\Lambda_{k-1} = I - \rho_{k-1} y_{k-1} s_{k-1}^T. \tag{۶.۲}$$

با اینکه BFGS به‌جای محاسبه $\nabla^2 f_k$ ، تقریبی از آن را به‌دست می‌آورد، اما باز هم هزینه محاسبه و ذخیره این تقریب زیاد است. به‌خصوص وقتی تعداد متغیرها زیاد باشد. برای حل این مشکل، نسخه حافظه محدود این الگوریتم با نام L–BFGS ارائه شد. در L–BFGS به‌جای ذخیره صریح H_k، تعدادی بردار نگهداری می‌کنیم که همان جفت بردارهای $\{y_i, s_i\}$ هستند. این جفت بردارها نقشی کلیدی در به‌روزرسانی ماتریس هسی تقریبی دارند. تعداد بردارهای ذخیره‌شده، توسط یک پارامتر به نام m مشخص می‌شود که نشان‌دهنده‌ی طول حافظه الگوریتم است. بردارهای ذخیره‌شده توسط L–BFGS برای ساخت و تقریب ضمنی H_k استفاده می‌شوند. سپس مقدار $H_k \nabla f_k$ را می‌توان با انجام یک سری ضرب‌های داخلی و جمع‌های برداری به‌دست آورد. دقت کنید که در L–BFGS مقدار $H_k \nabla f_k$ محاسبه می‌شود و نه مقدار H_k. برای درک بهتر ماهیت تکراری بودن L–BFGS و با کمک فرمول (۴.۲) می‌توان فرمول (۷.۲) را به‌دست آورد که در آن H_k با یک سری عملیات مشابه و تکراری در هر تکرار به‌دست می‌آید:

$$\begin{aligned} H_k =& (\Lambda_{k-1}^T \cdots \Lambda_{k-m}^T) H_k^{\circ} (\Lambda_{k-m} \cdots \Lambda_{k-1}) \\ &+ \rho_{k-m} (\Lambda_{k-1}^T \cdots \Lambda_{k-m+1}^T) s_{k-m} s_{k-m}^T (\Lambda_{k-m+1} \cdots \Lambda_{k-1}) \\ &+ \rho_{k-m+1} (\Lambda_{k-1}^T \cdots \Lambda_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (\Lambda_{k-m+2} \cdots \Lambda_{k-1}) \\ &+ \cdots + \rho_{k-1} s_{k-1} s_{k-1}^T, \end{aligned} \tag{۷.۲}$$

الگوریتم ۱ الگوریتم دو حلقه بازگشتی L–BFGS

۱: $q \leftarrow \nabla f_k$
۲: تکرار خطوط ۳ و ۴ به ازای $i = k-1$ تا $i = k-m$
۳: $\alpha_i \leftarrow \rho_i s_i^T q$
۴: $q \leftarrow q - \alpha_i y_i$
۵: $r \leftarrow H_k^\circ q$
۶: تکرار خطوط ۷ و ۸ به ازای $i = k-m$ تا $i = k-1$
۷: $\beta \leftarrow \rho_i y_i^T r$
۸: $r \leftarrow r + s_i(\alpha_i - \beta)$
۹: $H_k \nabla f_k = r$

الگوریتم ۲ L–BFGS

۱: انتخاب $w_\circ$
۲: انتخاب $m > \circ$
۳: $k \leftarrow \circ$
۴: تکرار خطوط ۵ تا ۱۱ تا زمان برقراری شرط توقف
۵: انتخاب H_k°
۶: $p_k \leftarrow -H_k \nabla f_k$ از الگوریتم ۱
۷: $w_{k+1} \leftarrow w_k + \alpha_k p_k$ (α_k طوری انتخاب میشود که شرایط وُلف[۵] برقرار باشد)
۸: اگر $k > m$ است s_{k-m}, y_{k-m} حذف شود
۹: $s_k \leftarrow w_{k+1} - w_k$
۱۰: $y_k \leftarrow \nabla f_{k+1} - \nabla f_k$
۱۱: $k \leftarrow k+1$

در این معادله مقدار اولیه H_k° را می‌توان

$$H_k^\circ = \gamma_k I, \tag{۸.۲}$$

در نظر گرفت. که در این عبارت

$$\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}. \tag{۹.۲}$$

با کمک فرمول (۷.۲)، الگوریتم تکرار دو حلقه L–BFGS به‌دست می‌آید (الگوریتم ۱). در این الگوریتم دو حلقه وجود دارد. هر حلقه به اندازه طول حافظه (m) تکرار می‌شود. هر دو حلقه محاسباتشان به‌صورت سریالی انجام می‌شود. یعنی هر تکرار برای محاسبات خود، نیاز به خروجی تکرار قبلی دارد. مشکل الگوریتم‌های سریالی زمان‌بر بودن آنها و عدم قابلیت موازی‌سازی است. بعد از به‌دست آوردن $H_k \nabla f_k$ و همچنین تعیین طول گام (α_k) مقدار جدید w_{k+1} به‌دست می‌آید و طبق آن مقادیر جدید s_k و y_k محاسبه می‌شود و با تکرار این روند تا زمان برقراری شرط توقف، الگوریتم L–BFGS به‌دست می‌آید (الگوریتم ۲). شرط توقف می‌تواند براساس نرم گرادیان تابع f در گام k و یا میزان تغییرات w و یا $f(w)$ در دو گام متوالی تعیین شود. همچنین در کارهای انجام شده در حوزه یادگیری ماشین شرط توقف می‌تواند رسیدن به مقدار مشخص برای تابع هدف باشد. درکارهای یادگیری ماشین چون تابع هدف اغلب مبتنی بر تابع زیان تعریف می‌شود، می‌توانیم بگوییم برای مثال اگر مقدار تابع هدف کمتر از ۰/۰۰۰۱ شد، اجرا متوقف شود. همچنین در برخی موارد، شرط توقف می‌تواند براساس یک محدودیت زمانی و یا محدودیت در تعداد تکرارها تعیین شود.

روش‌های زیادی در جهت تعمیم و بهبود L–BFGS و BFGS و همچنین تحلیل این الگوریتم‌ها ارائه شده‌اند [۸، ۱۰]. L–BFGS و BFGS هر دو از گرادیان استفاده می‌کنند. برخی روش‌های ارائه شده نظیر BFGS برخط [۱۱] سعی دارند الگوریتم BFGS را به گونه‌ای تغییر دهند تا با گرادیان تصادفی نیز به‌خوبی کار کند. نسخه‌های دیگری از BFGS برخط نظیر BFGS تصادفی منظم شده [۱۲] که با نام RES هم شناخته می‌شود نیز ارائه شده که پایداری بیشتری نسبت به نسخه برخط اولیه دارد.

برخی مقالات روی بهبود نرخ همگرایی BFGS کار کرده‌اند و توانستند به نرخ همگرایی فوق خطی برسند [۱۳، ۱۴] که البته این همگرایی یا از نوع محلی است و یا روی توابع قویاً محدب آزمایش شده است [۱۵]. نسخه‌های برخطی برای L–BFGS نیز ارائه شده است

[۵] برای مثال، بنگرید به کتاب [۱]، فصل ۳، صفحات ۳۳ تا ۳۶.

[۱۶]. برخی از این روش‌ها از تکنیک‌های کاهش واریانسی که در روش‌های مرتبه اول [۱۷–۱۹] وجود دارند استفاده می‌کنند تا به نرخ همگرایی بهتری دست یابند. برای مثال در [۹] استفاده از روش‌های کاهش واریانس ارائه شده در [۱۸] منجر به کم‌شدن نویز گرادیان تخمینی شده است که دستاوردش نرخ همگرایی خطی است. روش‌های دیگری نیز در همین راستا ارائه شدند که هر کدام به نوعی سعی در کاهش نویز گرادیان و بهبود نرخ همگرایی دارند [۲۰–۲۲]. در این بین برخی به نرخ همگرایی فوق خطی محلی نیز دست پیدا کرده‌اند [۲۳–۲۵]. هرچند اغلب این روش‌ها در داده‌های با ابعاد بالا، کاربردی نیستند.

علاوه‌بر مبحث کاهش نویز، برخی مقالات روی نمایش فشرده ماتریس هسی کار کرده‌اند [۲۶، ۲۷]. مقالاتی نیز در حوزه توابع ناهموار و البته محدب کار می‌کنند [۲۸]. با توجه به اهمیت حوزه یادگیری ماشین [۲۹، ۳۰] و آموزش دسته‌ای الگوریتم‌ها در این حوزه، برخی مقالات روی موازی کردن و توزیع پردازش دسته‌ها تمرکز کرده‌اند [۳۱]. همچنین کار روی توابع هزینه غیرمحدب و بهینه‌سازی آنها با روش‌های شبه‌نیوتن از دیگر حوزه‌های مهمی است که کارهای زیادی روی آن انجام شده است [۲۱، ۲۲، ۳۲–۳۴]. روش‌های مختلفی هم برای تعیین اندازه مناسب حافظه در L–BFGS ارائه شده است [۳۵]. به‌طور کلی تعمیم‌ها و بهبودهای زیادی روی دو الگوریتم BFGS و L–BFGS ارائه شده است که نشان می‌دهد این الگوریتم‌ها از اهمیت زیادی برخوردارند.

۳ روش ارائه شده

الگوریتم دو حلقه L–BFGS نیازمند محاسبات سریالی است که منجر به کاهش سرعت اجرا و عدم توانایی انجام محاسبات به‌صورت موازی می‌شود. هدف ما حذف این محاسبات سریالی و جایگزینی آنها با محاسباتی است که بتوان آنها را به‌صورت موازی نیز اجرا کرد. بدین منظور ما از دیدگاه عملگری به مساله نگاه می‌کنیم. در این دیدگاه ماتریس H یک عملگر است که گرادیان ورودی را دریافت و یک خروجی به‌دست می‌دهد. ذخیره H به‌صورت صریح هزینه محاسباتی و حافظه مصرفی زیادی از مرتبه دو دارد. بنابراین برای ذخیره‌سازی و محاسبه این عملگر در هر تکرار، یک تقریب رتبه پایین از H به‌دست آوریم. طبق این رویکرد نیازی به محاسبات سریالی در تقریب ماتریس هسی نیست. همچنین الگوریتم ما مانند L–BFGS توانایی کار با حافظه محدود m را دارد و رشد میزان حافظه مصرفی نیز نسبت به d از مرتبه خطی است. طبق آزمایش‌های ما اعمال این رویکرد می‌تواند در برخی مسائل نظیر مساله درجه دو سرعت اجرا را تا ده برابر نسبت به L–BFGS افزایش دهد. در ادامه ابتدا یک تقریب رتبه پایین برای ماتریس H_k معرفی می‌کنیم. سپس مقدار $H_k\nabla f_k$ را بر اساس این تقریب رتبه پایین به‌دست می‌آوریم.

ماتریس H به عنوان یک عملگر، در هر تکرار نیاز به به‌روزرسانی دارد. یعنی عملگر در لحظه k متفاوت با عملگر در لحظه $k+1$ است. بنابراین در هر تکرار باید یک تقریب رتبه پایین برای H_k به‌دست آوریم. فرض کنید طبق معادله (۴.۱) می‌توانیم H_{k-1} را تجزیه کنیم:

$$H_{k-1} = \sum_{i=1}^{r} \sigma_i u_i v_i^T. \tag{۱.۳}$$

در این معادله u_i و v_i بردارهایی d بعدی و σ_i یک اسکالر است. r هم تعداد بردارها و مقادیر لازم جهت تقریب رتبه پایین H_{k-1} است. مقدار r برابر $m \times b$ است که b یک ثابت عددی است. مقدار b به نوع پیاده‌سازی بستگی دارد ولی در روش ما این عدد برابر ۳ است. با باز کردن مقدار r می‌توان معادله (۱.۳) را به شکل زیر بازنویسی کرد:

$$H_{k-1} = \sum_{i=k-m}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} u_{i,j} v_{i,j}^T. \tag{۲.۳}$$

با توجه به اینکه روش ما یک روش حافظه محدود است، اگر تعداد عناصر ذخیره‌شده در حافظه به m رسیده باشد، آنگاه قدیمی‌ترین عناصر از حافظه حذف می‌شوند و عناصر جدید در حافظه ذخیره می‌شوند. با توجه به این توضیح ما برای به‌دست آوردن H_k مستقیما از H_{k-1} استفاده نمی‌کنیم، بلکه از $\hat{H}_{k-1}$ استفاده می‌کنیم که به‌صورت زیر تعریف می‌شود:

$$\hat{H}_{k-1} = \sum_{i=k-m+1}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} u_{i,j} v_{i,j}^T. \tag{۳.۳}$$

در (۳.۳) مقادیر لحظه $i = k-m$ حذف شده‌اند. اکنون با کمک فرمول زیر، مقدار H_k را بر اساس $\hat{H}_{k-1}$ به‌دست می‌آوریم:

$$H_k = \Lambda_{k-1}^T \hat{H}_{k-1} \Lambda_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^T. \tag{۴.۳}$$

سپس با جایگذاری مقدار $\hat{H}_{k-1}$ از معادله (۳.۳) در (۴.۳) به معادله (۵.۳) و با انجام ضرب‌ها و فاکتورگیری‌ها به معادله (۶.۳) می‌رسیم.

$$\begin{aligned} H_k =& \rho_{k-1} s_{k-1} s_{k-1}^T + (I - \rho_{k-1} s_{k-1} y_{k-1}^T) \\ & (\sum_{i=k-m+1}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} u_{i,j} v_{i,j}^T)(I - \rho_{k-1} y_{k-1} s_{k-1}^T), \end{aligned} \tag{۵.۳}$$

$$\begin{aligned} H_k =& \sum_{i=k-m}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} u_{i,j} v_{i,j}^T \\ & - \rho_{k-1} s_{k-1} \sum_{i=k-m+1}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} y_{k-1}^T u_{i,j} v_{i,j}^T \\ & - \rho_{k-1} \sum_{i=k-m+1}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} v_{i,j}^T y_{k-1} u_{i,j} s_{k-1}^T \\ & + \rho_{k-1}(1 + \rho_{k-1} \sum_{i=k-m+1}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} y_{k-1}^T u_{i,j} v_{i,j}^T y_{k-1}) s_{k-1} s_{k-1}^T. \end{aligned} \tag{۶.۳}$$

براساس معادله (۶.۳) می‌توانیم به‌جای محاسبه صریح H_k، آن را براساس یک سری بردار و اسکالر نشان دهیم و ذخیره کنیم. این عبارت حاصل جمع ۴ جمله است. جمله اول حاصل جمع جملات از لحظات قبل است و هیچ متغیر جدیدی نظیر y_{k-1} یا s_{k-1} در آن وجود ندارد. سه جمله دیگر را می‌توان در قالب ۳ جفت بردار $(u_{k,1}, v_{k,1}), (u_{k,2}, v_{k,2}), (u_{k,3}, v_{k,3})$ و ۳ اسکالر $\sigma_{k,1}, \sigma_{k,2}, \sigma_{k,3}$ به‌صورت جدول ۱ ذخیره کرد. در این حالت $b = 3$. همچنین در گام ابتدایی $k = 0$ مقدار اولیه بردارهای $u_{i,j}$ و $v_{i,j}$ و مقادیر $\sigma_{i,j}$ را به‌صورت زیر تعریف می‌کنیم:

$$\sigma_{i,j} = 0 \quad \text{and} \quad v_{i,j} = u_{i,j} = 0 \quad \text{if} \quad i \leq 0 \tag{۷.۳}$$

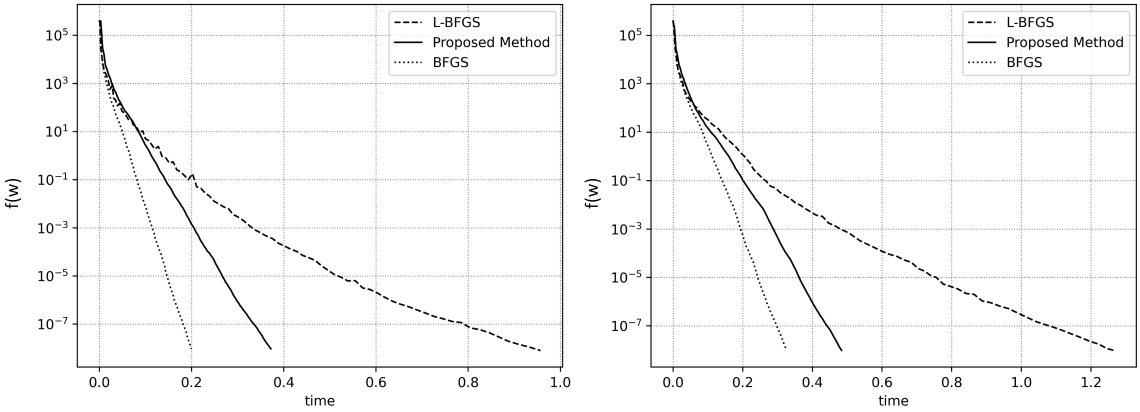
مقدار اولیه برای ماتریس H را هم برابر $(s_0^T y_0 / y_0^T y_0) \times I$ قرار می‌دهیم. نحوه تعیین مقدار اولیه و تعیین بردارهای $u_{i,j}$ و $v_{i,j}$ به گونه‌ای است که ماتریس H_k متقارن باشد. با ذخیره‌سازی H_k به‌صورت رتبه پایین، می‌توانیم مقدار $H_k \nabla f_k$ را به‌صورت زیر محاسبه کنیم:

$$H_k \nabla f_k = \sum_{i=k-m+1}^{k} \sum_{j=1}^{b} \sigma_{i,j} u_{i,j} v_{i,j}^T \nabla f_k \tag{۸.۳}$$

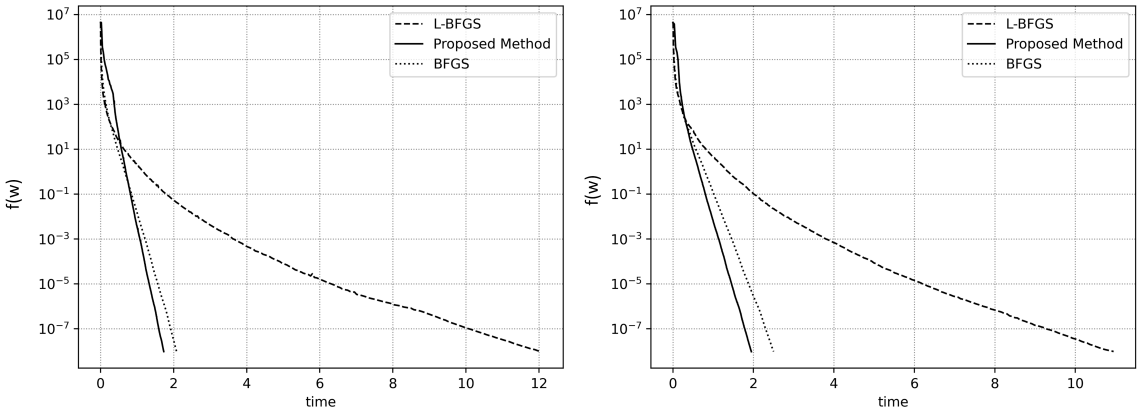
الگوریتم ارائه شده توسط ما نیاز به محاسبات سریالی ندارد و در هر تکرار k فقط با ضرب بردارها و اسکالرهای تقریب رتبه پایین در گرادیان، مقدار $H_k \nabla f_k$ را به‌دست می‌آورد. میزان حافظه موردنیاز برای الگوریتم ما از مرتبه $O(mbd)$ است.

جدول ۱: مقادیر $\sigma_{k,j}$، $u_{k,j}$ و $v_{k,j}$ به ازای $j = 1, 2, 3$.

$\sigma_{k,1}$	$-\rho_{k-1}$
$u_{k,1}$	s_{k-1}
$v_{k,1}$	$\sum_{i=k-m+1}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} y_{k-1}^T u_{i,j} v_{i,j}$
$\sigma_{k,2}$	$-\rho_{k-1}$
$u_{k,2}$	$\sum_{i=k-m+1}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} v_{i,j}^T y_{k-1} u_{i,j}$
$v_{k,2}$	s_{k-1}
$\sigma_{k,3}$	$\rho_{k-1}(1 + \rho_{k-1} \sum_{i=k-m+1}^{k-1} \sum_{j=1}^{b} \sigma_{i,j} y_{k-1}^T u_{i,j} v_{i,j}^T y_{k-1})$
$u_{k,3}$	s_{k-1}
$v_{k,3}$	s_{k-1}



شکل ۱: نتایج روی مساله درجه دو ۵۰۰ بعدی. شکل راست با جستجوی خطی و شکل چپ بدون جستجوی خطی است.



شکل ۲: نتایج روی مساله درجه دو ۲۰۰۰ بعدی. شکل راست با جستجوی خطی و شکل چپ بدون جستجوی خطی است.

۴ آزمایش‌ها و نتایج

در این بخش، ما روش خود را با L–BFGS و BFGS مقایسه می‌کنیم. پیاده‌سازی L–BFGS در PyTorch [۳۶] آمده است و BFGS را هم بر همین اساس پیاده‌سازی می‌کنیم. PyTorch یک کتابخانه یادگیری ماشین منبع باز است که براساس زبان برنامه‌نویسی پایتون و کتابخانه Torch توسعه داده شده است. تمام آزمایش‌ها روی سخت‌افزاری با ۱۶ گیگابایت رم و یک پردازنده با فرکانس ۴/۲ گیگاهرتز اجرا شده است و پردازنده گرافیکی مورد استفاده هم GTX۱۰۸۰ با ۸۱۹۲ مگابایت حافظه است.

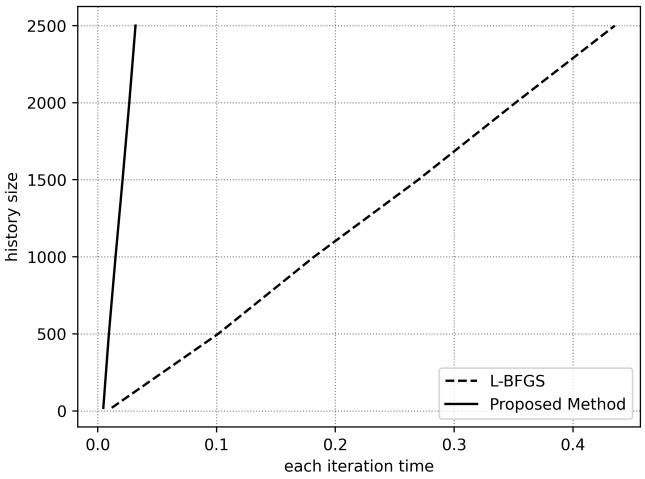
۱.۴ مساله درجه دو

تابع هدف در این آزمایش به‌صورت زیر است:

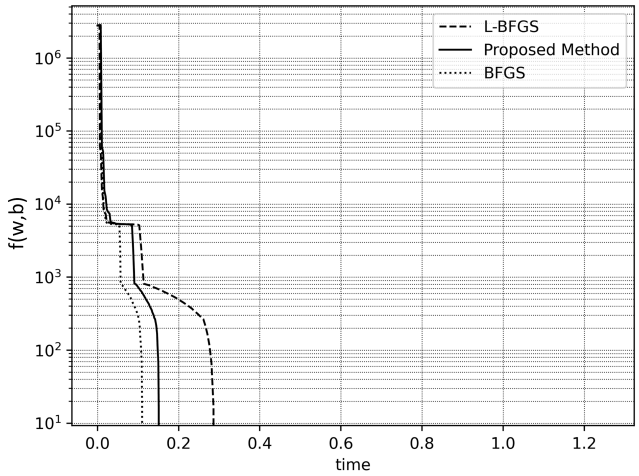
$$f(w) = \|Aw - b\|^2. \tag{۱.۴}$$

در عبارت بالا A یک ماتریس n در d است که $d < n$. همچنین b یک بردار n در ۱ است. این ماتریس و بردار را به صورت تصادفی و با کمک تابع randn در Pytorch ایجاد می‌کنیم. این تابع از یک توزیع نرمال با میانگین ۰ و انحراف معیار ۱ برای نمونه برداری استفاده می‌کند. هدف به دست اوردن مجموعه پارامتر $w_{d\times 1}$ مناسب برای این مساله است. در اینجا روی دو مساله ۵۰۰ و ۲۰۰۰ بعدی آزمایش می‌کنیم. در مساله ۵۰۰ بعدی $d = ۵۰۰$ و $n = ۷۰۰$ و در مساله ۲۰۰۰ بعدی $d = ۲۰۰۰$ و $n = ۲۲۰۰$ است. طول گام را برای تمام بهینه‌سازها ۱ در نظر می‌گیریم. برای مساله ۵۰۰ بعدی طول حافظه L–BFGS و روش ارائه شده را برابر ۱۰۰ و برای ۲۰۰۰ بعدی برابر ۴۰۰ در نظر می‌گیریم. همچنین آزمایش‌ها را در دو حالت با و بدون جستجوی خطی انجام می‌دهیم.

نتایج ۵۰۰ بعدی در شکل ۱ و نتایج ۲۰۰۰ بعدی در شکل ۲ آمده است. در تمام آزمایش‌ها هر سه روش به نقطه بهینه با تقریب بسیار خوبی رسیده‌اند. در مساله ۵۰۰ بعدی BFGS از نظر زمانی عملکرد بهتری نسبت به دو روش دیگر داشته است. اما در مساله ۲۰۰۰



شکل ۳: زمان اجرای هر تکرار



شکل ۴: نتیجه آزمایش تابع زیان لولا به ازای مجموعه داده ۲۰۰۰ نمونه‌ای ۱۰ بعدی.

بعدی روش ما بهتر عمل کرده است. همچنین در تمام آزمایش‌ها L–BFGS با اختلاف زیادی بیشترین زمان اجرا را به خود اختصاص داده است.

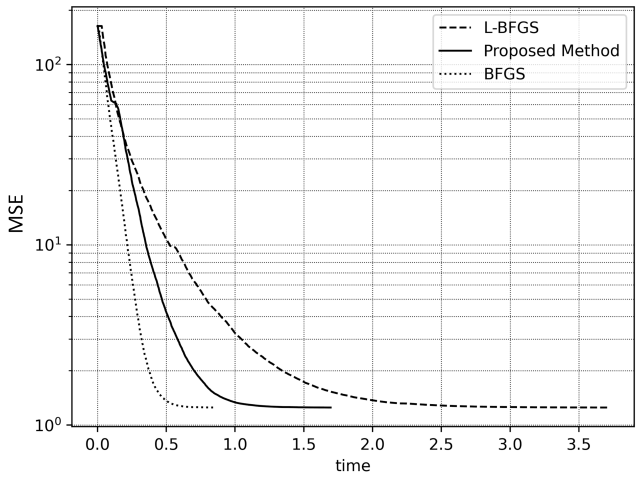
برای مقایسه دقیق‌ترِ سرعتِ اجرای روش خود با L–BFGS آزمایشی روی داده ۲۰۰۰ بعدی طراحی می‌کنیم که در آن اندازه حافظه را ۲۵۰۰ در نظر می‌گیریم. نتایج در شکل ۳ آمده است. در این شکل زمان اجرا در هر تکرار آمده است. برای مثال در تکرار ۱۵۰۰ ، روش ما حدود ۰/۰۲۷ ثانیه و در L–BFGS حدود ۰/۲۷ ثانیه زمان می‌برد. یعنی روش ما در این حالت ۱۰ برابر از L–BFGS سریع‌تر است.

۲.۴ تابع زیان لولا (**Hinge loss**)

در این بخش برای ارزیابی و مقایسه روش ارائه شده با BFGS و L–BFGS از تابع زیان لولا استفاده می‌کنیم:

$$l^{hinge}(z) = \max(0, 1 - z). \tag{۲.۴}$$

در اینجا یک مجموعه داده طراحی می‌کنیم که x_i ویژگی داده شماره i و t_i برچسب آن داده است. این مجموعه شامل $n = 2000$ داده $d = 10$ بعدی است. همچنین این مجموعه دارای دو دسته داده است که به صورت خطی قابل تفکیک هستند. براساس توضیحات بالا و همچنین تابع زیان لولا، تابع هدف را به صورت زیر تعریف می‌کنیم:



شکل ۵: رگرسیون خطی روی مجموعه داده California housing

$$f(w,b) = \sum_{i=1}^{n} l^{hinge}(t_i(w^T x_i - b)). \tag{۳.۴}$$

در عبارت بالا w و b پارامترهای مدل هستند و درنتیجه $w^T x_i - b$ خروجی مدل به ازای ورودی x_i است. در این آزمایش طول حافظه L–BFGS و روش خودمان را ۳۰۰ در نظر می‌گیریم. همچنین طول گام را برای هز سه روش، ۱ در نظر می‌گیریم. نتایج در شکل ۴ قابل مشاهده است. روش ما نسبت به L–BFGS در زمان کمتری به پاسخ رسیده است اما نسب به BFGS کمی کندتر عمل کرده است.

۳.۴ رگرسیون خطی

در این بخش از مجموعه داده California housing استفاده می‌کنیم که شامل ۲۰۶۴۰ داده ۸ بعدی است. همچنین از خطای میانگین مربعات (MSE[۶]) به عنوان معیار ارزیابی استفاده می‌کنیم. طول حافظه را برای هر دو روش ارائه شده و L–BFGS برابر ۲۰۰ در نظر می‌گیریم. همچنین طول گام را برای هر سه روش ارائه شده، BFGS و L–BFGS برابر ۰/۰۱ قرار می‌دهیم. نتیجه رگرسیون در شکل ۵ آمده است. در این آزمایش BFGS از نظر زمانی بهتر عمل کرده است و در زمان ۰/۸۵ ثانیه به زیان کمینه رسیده است. پس از آن روش ما با زمان ۱/۷ و L–BFGS با ۳/۷ به زیان کمینه رسیده‌اند.

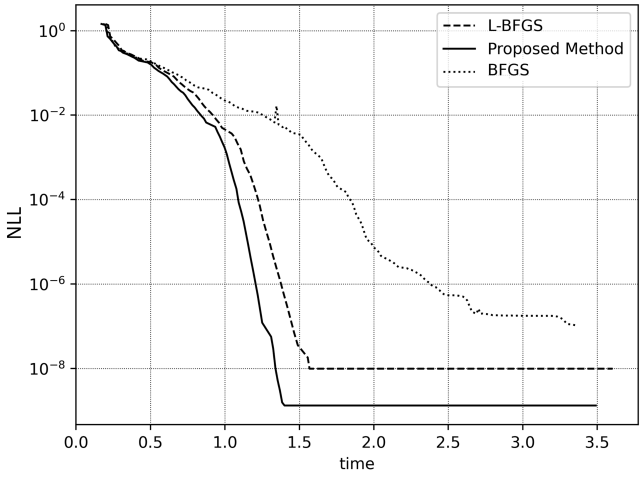
۴.۴ طبقه‌بندی داده‌ها با شبکه‌های عصبی مصنوعی

در این بخش می‌خواهیم داده‌ها را با شبکه‌های عصبی مصنوعی طبقه‌بندی کنیم و برای بهینه‌سازی از روش ارائه شده، L–BFGS و BFGS استفاده می‌کنیم. هدف ما در اینجا ارزیابی روش ارائه شده در حالت گرادیان تصادفی است. آزمایش‌ها روی دو مجموعه داده انجام می‌شود که هر کدام را در ادامه در بخشی جداگانه مورد بررسی قرار می‌دهیم. معیار ارزیابی و همچنین معیار آموزش شبکه در هر دو آزمایش منفی لگاریتم درست‌نمایی (NLL[۷]) است.

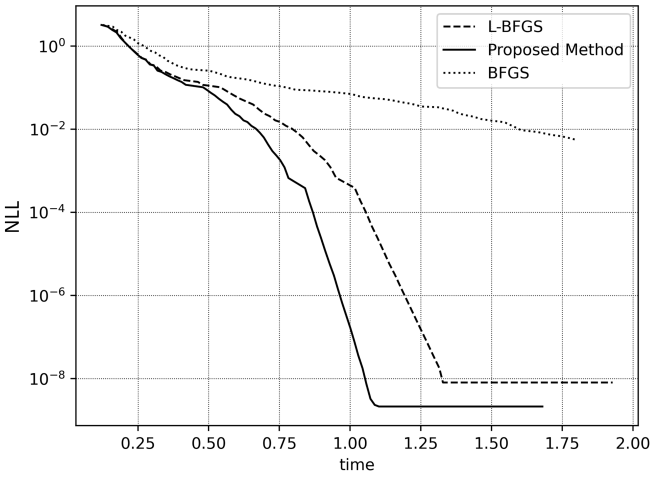
۱.۴.۴ مجموعه داده مصنوعی

این مجموعه ۲۰۰۰ داده ۲۰ بعدی و ۴ کلاس دارد. شبکه‌ای که برای طبقه‌بندی این داده‌ها در نظر گرفتیم دولایه کاملاً متصل دارد و شبکه در مجموع شامل $d = ۳۷۹$ پارامتر است. در روش ارائه شده طول حافظه را ۲۵۰ و برای L–BFGS طول حافظه را ۱۰۰ در

[۶] Mean Squared Error
[۷] Negative Log-Likelihood



شکل ۶: طبقه‌بندی روی مجموعه داده مصنوعی



شکل ۷: طبقه‌بندی روی اعداد دست‌نویس UCI

نظر می‌گیریم. طول گام برای هر دو روش ۱ در نظر گرفته شده است. نتایج در شکل ۶ آمده است. روش ارائه شده و L–BFGS به‌دلیل سادگی مساله، به‌خوبی زیان را کاهش داده‌اند. اما روش BFGS به‌دلیل اینکه نیاز به تشکیل ماتریس هسی دارد که شامل محاسباتی از مرتبه $O(d^2)$ است، بسیار کند عمل کرده است و نتوانسته از نظر زمانی به خوبی دو الگوریتم دیگر عمل‌کند.

روش ارائه شده با اینکه طول حافظه بیشتری نسبت به L–BFGS دارد، باز هم از نظر سرعت بهتر از L–BFGS عمل کرده است. روش ما پس از حدود ۱/۴ ثانیه زیان را به 10^{-8} رسانده است، درحالی‌که L–BFGS حدود ۱/۵ ثانیه زمان نیاز دارد تا به این مقدار برسد.

۲.۴.۴ مجموعه داده اعداد دست‌نویس UCI

این مجموعه داده شامل ۱۷۹۷ داده است و هر داده یک تصویر ۸ در ۸ از یک رقم دست‌نویس است. تعداد کلاس‌ها ۱۰ تا است که نشان‌گر اعداد ۰ تا ۹ هستند. شبکه عصبی در نظر گرفته شده برای این مساله مانند مساله قبل دو لایه کاملاً متصل دارد. اما این بار تعداد پارامترها برابر $d = 1510$ است. روش ارائه شده و L–BFGS طول حافظه ۱۰۰ دارند و همچنین در هر سه روش ارائه شده، BFGS و L–BFGS طول گام را ۱ در نظر گرفتیم. در این آزمایش که نتیجه آن در شکل ۷ قابل مشاهده است، روش ما عملکرد بهتری در سرعت از خود نشان داده است و در زمان کمتری نسبت به L–BFGS به زیان 10^{-8} رسیده است. همچنین روش BFGS به دلیل تعداد زیاد پارامترها و بزرگ بودن ابعاد مسأله، عملکرد کندی داشته است؛ چرا که این روش نیاز به تقریب و نگهداری صریح ماتریس هسی دارد که با افزایش ابعاد، هزینه

محاسباتی آن به شدت بالا می‌رود.

۵.۴ تحلیل کلی نتایج

روش ارائه شده در تمامی آزمایش‌ها عملکرد سریع‌تری نسبت به L–BFGS از خود نشان داد. همچنین، در مسئله درجه دوم (شکل ۳) مشاهده می‌شود که با افزایش طول حافظه، زمان اجرای هر دو روش به‌صورت خطی افزایش می‌یابد؛ با این حال، نرخ افزایش زمان در روش پیشنهادی تقریباً ۱۰ برابر کمتر از L–BFGS است. روش ارائه شده در مقایسه با BFGS در برخی آزمایشها سریع‌تر و در برخی کندتر عمل کرده است. با توجه به اینکه BFGS مستلزم تقریب و ذخیره‌سازی ماتریس هسین به‌صورت صریح است، هزینه محاسباتی آن در هر تکرار از مرتبه $O(d^2)$ نسبت به تعداد پارامترها (d) است. اما محاسبات روش ما در هر تکرار از مرتبه $O(d)$ است. بنابراین در آزمایشهایی که d مقدار بزرگی است، روش ما بهتر عمل کرده است. نظیر مساله درجه دو ۲۰۰۰ بعدی و آزمایشهای طبقه‌بندی در بخش ۴.۴.

۵ جمع‌بندی و کارهای آتی

هدف ما ارائه یک بهینه‌ساز جدید از دیدگاه عملگرِ ماتریس بود. عملگری که گرادیان را دریافت و خروجی جدیدی جهت به‌روزرسانی پارامترها برمی‌گرداند. در روش ارائه شده هیچ محاسبه سریالی وجود ندارد. عدم وجود محاسبات سریالی منجر به قابلیت موازی‌سازی محاسبات می‌شود که ما در روش خود از آن بهره بردیم. افزون‌بر این به‌دلیل استفاده از یک نمایش رتبه پایین، روش ما نسبت به تعداد متغیرها، رشد حافظه خطی دارد. روش ما در بعضی از موارد عملکرد خوبی ندارد. در حوزه گرادیان تصادفی باید روی کاهش نویز گرادیان و عملکرد بهتر در حالت دسته‌ای کار کنیم. افزون‌بر این روش ما در حالتی که طول حافظه کم باشد عملکرد خوبی ندارد و در این حوزه هم نیاز به بهبود دارد.

فهرست منابع

[1] Nocedal, J. and Wright, S. J., 2006. *Numerical optimization*. 2nd ed. New York: Springer. doi: 10.1007/978-0-387-40065-5

[2] Broyden, C. G., 1970. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, *6*(1), pp. 76-90. doi: 10.1093/imamat/6.1.76

[3] Fletcher, R., 1970. A new approach to variable metric algorithms. *The Computer Journal*, *13*(3), pp. 317-322. doi: 10.1093/comjnl/13.3.317

[4] Goldfarb, D., 1970. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, *24*(109), pp. 23-26. doi: 10.2307/2004873

[5] Shanno, D. F., 1970. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, *24*(111), pp. 647-656. doi: 10.2307/2004840

[6] Nocedal, J., 1980. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, *35*(151), pp. 773-782. doi: 10.2307/2006193

[7] Liu, D. C. and Nocedal, J., 1989. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, *45*(1), pp. 503-528. doi: 10.1007/BF01589116

[8] Mokhtari, A. and Ribeiro, A., 2020. Stochastic Quasi-Newton Methods. *Proceedings of The IEEE*, *108*(11), pp. 1906-1922. doi: 10.1109/JPROC.2020.3023660

[9] Moritz, P., Nishihara, R. and Jordan, M., 2016. A linearly-convergent stochastic L-BFGS algorithm. *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 249-258.

[10] Xie, Y., Byrd, R. H. and Nocedal, J., 2020. Analysis of the BFGS method with errors. *SIAM Journal on Optimization*, *30*(1), pp. 182-209. doi: 10.1137/19M1240794

[11] Bordes, A., Bottou, L. and Gallinari, P., 2009. SGD-QN: Careful quasi-Newton stochastic gradient descent. *Journal of Machine Learning Research*, *10*, pp. 1737-1754. doi: 10.5555/1577069.1755842

[12] Mokhtari, A. and Ribeiro, A., 2014. RES: Regularized stochastic BFGS algorithm. *IEEE Transactions on Signal Processing*, *62*(23), pp. 6089-6104. doi: 10.1109/TSP.2014.2357775

[13] Gao, W. and Goldfarb, D., 2019. Quasi-Newton methods: superlinear convergence without line searches for self-concordant functions. *Optimization Methods and Software*, *34*(1), pp. 194-217. doi: 10.1080/10556788.2018.1510927

[14] Rodomanov, A. and Nesterov, Y., 2021. Greedy quasi-Newton methods with explicit superlinear convergence. *SIAM Journal on Optimization*, *31*(1), pp. 785-811. doi: 10.1137/20M1320651

[15] Rodomanov, A. and Nesterov, Y., 2022. Rates of superlinear convergence for classical quasi-Newton methods. *Mathematical Programming*, pp. 1-32. doi: 10.1007/s10107-021-01622-5

[16] Schraudolph, N. N., Yu, J. and Günter, S., 2007, March. A stochastic quasi-Newton method for online convex optimization. *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, (pp. 436-443).

[17] Mokhtari, A., Gurbuzbalaban, M. and Ribeiro, A., 2018. Surpassing gradient descent provably: A cyclic incremental method with linear convergence rate. *SIAM Journal on Optimization*, *28*(2), pp. 1420-1447. doi: 10.1137/16M1101702

[18] Johnson, R. and Zhang, T., 2013. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in Neural Information Processing Systems*, *26*, pp. 315-323.

[19] Schmidt, M., Le roux, N. and Bach, F., 2017. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, *162*, pp. 83-112. doi: 10.1007/s10107-016-1030-6

[20] Chang, D., Sun, S. and Zhang, C., 2019. An Accelerated Linearly Convergent Stochastic L-BFGS Algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(11), pp. 3338-3346. doi: 10.1109/TNNLS.2019.2891088

[21] Wang, X., Ma, S., Goldfarb, D. and Liu, W., 2017. Stochastic quasi-Newton methods for nonconvex stochastic optimization. *SIAM Journal on Optimization*, *27*(2), pp. 927-956. doi: 10.1137/15M1053141

[22] Chen, H., Wu, H. C., Chan, S. C. and Lam, W. H., 2020. A Stochastic Quasi-Newton Method for Large-Scale Nonconvex Optimization with Applications. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(11), pp. 4776-4790. doi: 10.1109/TNNLS.2019.2957843

[23] Mokhtari, A., Eisen, M. and Ribeiro, A., 2018a. IQN: An incremental quasi-Newton method with local superlinear convergence rate. *SIAM Journal on Optimization*, *28*(2), pp. 1670-1698. doi: 10.1137/17M1122943

[24] Javad ebadi, M., Fahs, A., Fahs, H. and Dehghani, R., 2023. Competitive secant (BFGS) methods based on modified secant relations for unconstrained optimization. *Optimization*, *72*(7), pp. 1691-1706. doi: 10.1080/02331934.2022.2048381

[25] Gao, Z., Koppel, A. and Ribeiro, A., 2020. Incremental greedy BFGS: An incremental quasi-Newton method with explicit superlinear rate. *Proceedings of the 12th OPT Workshop on Optimization for Machine Learning, Advances in Neural Information Processing Systems,* (p. 47).

[26] Gower, R., Goldfarb, D. and Richtárik, P., 2016, June. Stochastic block BFGS: Squeezing more curvature out of data. In *Proceedings of the 33rd International Conference on Machine Learning*, (pp. 1869-1878).

[27] Brust, J. J., Marcia, R. F., Petra, C. G. and Saunders, M. A., 2022. Large-scale optimization with linear equality constraints using reduced compact representation. *SIAM Journal on Scientific Computing*, *44*(1), pp. A103-A127. doi: 10.1137/21M1393819

[28] Jalilzadeh, A., Nedić, A., Shanbhag, U. V. and Yousefian, F., 2022. A variable sample-size stochastic quasi-Newton method for smooth and nonsmooth stochastic convex optimization. *Mathematics of Operations Research*, *47*(1), pp. 690-719. doi: 10.1109/CDC.2018.8619209

[29] Rafati, J. and Marica, R. F., 2020. Quasi-Newton Optimization Methods for Deep Learning Applications. *Deep Learning Applications*, pp. 9-38. doi: 10.1007/978-981-15-1816-4_2

[30] Goldfarb, D., Ren, Y. and Bahamou, A., 2020. Practical quasi-Newton methods for training deep neural networks. *Advances in Neural Information Processing Systems*, *33*, pp. 2386-2396.

[31] Berahas, A. S., Nocedal, J. and Takac, M., 2016. A multi-batch L-BFGS method for machine learning. *Advances in Neural Information Processing Systems*, *29*, pp. 1063-1071.

[32] Yang, M., Milzarek, A., Wen, Z. and Zhang, T., 2022. A stochastic extra-step quasi-Newton method for nonsmooth nonconvex optimization. *Mathematical Programming*, pp. 1-47. doi: 10.1007/s10107-021-01629-y

[33] Indrapriyadarsini, S., Mahboubi, S., Ninomiya, H. and Asai, H., 2019, September. A Stochastic Quasi-Newton Method with Nesterov's Accelerated Gradient. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 743-760). doi: 10.1007/978-3-030-46150-8_43

[34] Yousefian, F., Nedic, A. and Shanbhag, U. V., 2020. On stochastic and deterministic quasi-Newton methods for nonstrongly convex optimization: Asymptotic convergence and rate analysis. *SIAM Journal on Optimization*, *30*(2), pp. 1144-1172. doi: 10.1137/17M1152474

[35] Boggs, P. T. and Byrd, R. H., 2019. Adaptive, limited-memory BFGS algorithms for unconstrained optimization.*SIAM Journal on Optimization*, *29*(2), pp. 1282-1299. doi: 10.1137/16M1065100

[36] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A., 2017. Automatic differentiation in pytorch. *Advances in Neural Information Processing Systems*, pp. 8024–8035.

A Limited-Memory Quasi-Newton Optimization Method with Parallel Computing Capability

Ashkan Sadeghi-Lotfabadi[(1)] and Kamaledin Ghiasi-Shirazi[(1),8]

[(1)] Department of Computer Engineering, Engineering Faculty, Ferdowsi University of Mashhad, Mashhad, Iran

Communicated by: Mohammad Hadi Farahi

Received: 17 November 2024 Accepted: 28 May 2025

Abstract: Newton's method is considered preferable due to its higher accuracy compared to first-order methods in approximating the target function; however, it requires the explicit computation of the Hessian matrix. On the other hand, quasi-Newton methods are popular among researchers because they don't require directly calculating the Hessian matrix. Here, we propose a quasi-Newton method designed to achieve high computational speed while accurately approximating the Hessian. Our approach views the Hessian matrix from an operational perspective. From the operational perspective, a matrix acts as an operator that transforms an input vector into an output vector. Following this approach, the Hessian matrix takes a gradient vector as input and generates an output vector accordingly. Our method is designed to perform calculations in parallel, enabling high execution speed. Moreover, our method is specifically designed to operate efficiently with limited memory, making it well-suited for large-scale problems. Our experimental results indicate that the proposed method is faster than the quasi-Newton L-BFGS method. For instance, in applications such as quadratic optimization problems, our method demonstrates a significant performance improvement, executing approximately ten times faster than L-BFGS. Furthermore, in problems involving a large number of variables, our method is more computationally efficient than BFGS.

Keywords: Quasi-Newton, L-BFGS, operator, low-rank approximation.

[8]Corresponding author
E-mail addresses: (A. Sadeghi-Lotfabadi) sadeghia@mail.um.ac.ir, (K. Ghiasi-Shirazi) k.ghiasi@um.ac.ir