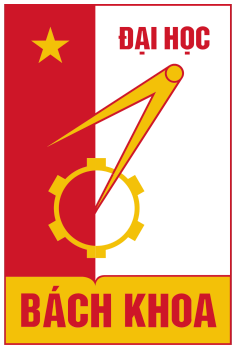
ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO BÀI TẬP LỚN

NHẬP MÔN CÁC PHƯƠNG PHÁP TỐI ƯU

Thuật toán tự thích nghi cho lập trình tựa lồi và ứng dụng trong học máy

Giảng viên hướng dẫn: **PGS. TS. Trịnh Ngọc Hải**
(Khoa Toán - Tin)

Danh sách sinh viên thực hiện:

Nguyễn Ngọc Tuấn Anh	– 202400029	Nguyễn Hồng Đạt	– 202416153
Trần Tuấn Anh	– 202416124	Phạm Đình Minh Đức	– 202400038
Nguyễn Hữu Chính	– 202416143	Bùi Tiến Dũng	– 202416167
Trần Mạnh Cường	– 202416147	Trần Huy Dương	– 20230025
Đỗ Hải Đăng	– 202400035	Nguyễn Khắc Duy	– 202400100
Bùi Tuấn Đạt	– 202400096	Nguyễn Hoàng Gia	– 202400040

HÀ NỘI, 12/2025

Mục lục

1 Giới thiệu

Các phương pháp gradient descent từ lâu đã đóng vai trò quan trọng trong việc giải quyết các bài toán tối ưu, từ những bài toán tối ưu lồi đến những bài toán phi lồi phức tạp, và được ứng dụng rộng rãi trong nhiều lĩnh vực như tối ưu, xử lý tín hiệu và học máy (xem [3], [4], [10]). Ở mỗi vòng lặp, thuật toán gradient descent tìm nghiệm tối ưu thông qua hướng gradient và cỡ bước. Trong nhiều năm, phần lớn nghiên cứu tập trung vào việc cải thiện hướng cập nhật nhằm tăng tốc độ hội tụ, trong khi lựa chọn cỡ bước lại chủ yếu dựa trên các phương pháp quen thuộc như tìm kiếm theo đường thẳng (line-search) hoặc sử dụng hằng số Lipschitz (xem [3], [14]).

Tuy nhiên, cùng với sự xuất hiện của nhiều lĩnh vực ứng dụng học máy mới với dữ liệu có số chiều rất lớn và hàm mục tiêu không lồi, nhu cầu về những chiến lược chọn cỡ bước hiệu quả, chi phí tính toán thấp ngày càng trở nên quan trọng (xem [4], [10]). Dù tìm chính xác hay gần đúng, các phương pháp tìm kiếm theo đường thẳng đòi hỏi chi phí tính toán lớn trong mỗi bước lặp, đặc biệt trong những trường hợp việc tính giá trị của hàm gần như tương đương với việc tính đạo hàm của nó và đòi hỏi phải giải quyết các bài toán phụ phức tạp [3]. Ngược lại, các kĩ thuật sử dụng giá trị xác định từ trước như hằng số Lipschitz để tính cỡ bước thường thiếu ổn định, dẫn đến tốc độ hội tụ chậm hoặc không tối ưu. Một số nghiên cứu khác, như quy tắc chuỗi phân kỳ, cũng có hạn chế tương tự (xem [8], [14]).

Bên cạnh đó, nhiều phương pháp mở rộng của phương pháp gradient dành cho các hàm tựa lồi, giả lồi hoặc các bài toán có ràng buộc phức tạp đã được đề xuất trước đây, nhưng hiệu quả còn hạn chế. Chẳng hạn như phương pháp giảm dần cỡ bước cổ điển trong bài toán tựa lồi [8] dẫn tới tốc độ hội tụ rất chậm; các phương pháp khác yêu cầu hàm mục tiêu thỏa điều kiện Hölder (xem [20], [7]), hoặc chỉ hoạt động với tập ràng buộc bị chặn. Ngoài ra, những mô hình dựa trên mạng nơ-ron hồi quy (RNN) cho các bài toán lập trình giả lồi có ràng buộc lại sử dụng cỡ bước cố định mà không có sự điều chỉnh (xem [2], [11]).

Một số nghiên cứu trước đây đã đề xuất các thuật toán tự điều chỉnh cỡ bước nhằm cải thiện hiệu quả của các phương pháp gradient descent. Các cách tiếp cận này thường hoạt động tốt đối với các bài toán giả lồi khi tập ràng buộc bị chặn [9], và đã có những thuật toán hiệu quả trong trường hợp tập ràng buộc không bị chặn [6]. Tuy nhiên, các phương pháp đó vẫn chưa áp dụng được cho những bài toán không ràng buộc, do còn tồn tại các giới hạn trong cách xây dựng và phân tích cỡ bước.

Từ những hạn chế trên, bài báo [17] đề xuất một thuật toán mới nhằm xây dựng cơ chế điều chỉnh cỡ bước tự thích nghi, không cần tìm kiếm theo đường thẳng và có khả năng áp dụng cho một lớp rộng các bài toán tối ưu có hàm mục tiêu không lồi, trơn, và tập ràng buộc lồi, đóng nhưng không bị chặn. Điểm then chốt của thuật toán là giảm dần cỡ bước một cách có kiểm soát cho đến khi thỏa mãn một điều kiện nhất định. Mặc dù tính liên tục Lipschitz của gradient của hàm mục tiêu là điều kiện cần cho sự hội tụ, thuật toán được đề xuất không sử dụng hằng số Lipschitz để tính toán cỡ bước. Thuật toán cũng bao gồm phép chiếu trực giao, đảm bảo nghiệm tìm được luôn nằm trong tập ràng buộc.

Một trong những đóng góp quan trọng nhất của bài báo [17] là mở rộng kĩ thuật điều chỉnh cỡ bước tự thích nghi sang trường hợp tập ràng buộc không bị chặn. Tuy nhiên, việc mở rộng này gặp phải một số khó khăn. Thứ nhất, cần đảm bảo

tính hội tụ của thuật toán mà không phải đưa thêm các điều kiện phụ trợ phức tạp. Các kết quả về sự hội tụ được chứng minh bằng cách khai thác các tính chất của hàm mục tiêu cùng với các biến đổi bất đẳng thức phù hợp. Thứ hai, mặc dù toán tử chiếu bảo đảm rằng điểm x^{k+1} được sinh ra từ x^k luôn nằm trong tập khả thi, ta vẫn phải chứng minh rằng sự xuất hiện của phép chiếu không làm ảnh hưởng đến tính hội tụ của thuật toán. Cuối cùng, thuật toán thích nghi được đề xuất phải bao hàm cả trường hợp sử dụng cỡ bước cố định. Trường hợp này cho thấy thuật toán là một phương pháp mở rộng từ phương pháp Gradient Descent truyền thống và rất hữu ích trong các ứng dụng thực tế, đặc biệt đối với những hàm mục tiêu không lồi. Các ví dụ tính toán trên những bài toán có quy mô lớn với hàm mục tiêu không lồi đã củng cố cho nhận định này.

Phần chứng minh lý thuyết trong bài báo [17] cho thấy dãy nghiệm sinh ra bởi thuật toán hội tụ về điểm dừng của bài toán; trong trường hợp hàm tựa lồi hoặc giả lồi, nghiệm thu được còn là nghiệm tối ưu. Nhìn chung, các kết quả trong bài báo [17] khẳng định rằng cơ chế điều chỉnh cỡ bước của thuật toán đủ tốt để đảm bảo tính ổn định và hội tụ ngay cả trong bối cảnh tối ưu phi lồi và tập ràng buộc không bị chặn.

Để đánh giá hiệu quả, các tác giả thực hiện nhiều thí nghiệm bao gồm các bài toán phi lồi kinh điển, các bài toán có ràng buộc phức tạp, và các bài toán quy mô lớn. Ngoài ra, thuật toán còn được ứng dụng vào nhiều bài toán học máy như chọn đặc trưng có giám sát, hồi quy logistic đa biến và huấn luyện mạng nơ-ron. Kết quả cho thấy phương pháp đề xuất có độ chính xác tốt, tốc độ tính toán vượt trội so với các thuật toán gradient descent và các mô hình thần kinh động học (RNN) trong cùng điều kiện.

Mục tiêu của báo cáo này là trình bày lại các ý chính và triển khai các thực nghiệm trong bài báo [17]. Phần tiếp theo của bài báo cáo gồm: Mục 2 giới thiệu cơ sở lý thuyết liên quan; Mục 3 mô tả thuật toán được đề xuất và chứng minh những kết quả quan trọng; Mục 4 phân tích các thực nghiệm; Mục 5 thảo luận các ứng dụng của thuật toán trong học máy; và Mục 6 đưa ra kết luận.

2 Cơ sở lý thuyết

Trong toàn bộ bài báo [17], ta giả định rằng C là một tập khác rỗng, đóng và lồi trong $\mathbb{R}^m$, $f : \mathbb{R}^m \to \mathbb{R}$ là một hàm khả vi trên một tập mở chứa C, ánh xạ ∇f liên tục Lipschitz, tức là tồn tại một hằng số $L > 0$ sao cho $||\nabla f(x) - \nabla f(y)|| \leq L||x - y||$ với mọi x, $y \in C$. Ta xét bài toán tối ưu:

$$\min_{x \in C} f(x). \qquad \text{(OP(f, C))}$$

Giả sử rằng tập nghiệm của OP(f, C) là không rỗng. Trước hết, ta nhắc lại một số định nghĩa và kết quả cơ bản sẽ được sử dụng trong phần tiếp theo của bài báo [17]. Chứng minh chi tiết được trình bày trong [1] và [15].

Với $x \in \mathbb{R}^m$, ký hiệu $P_C(x)$ là hình chiếu của x lên C, tức là:

$$P_C(x) := \text{argmin}\{||z - x|| : z \in C\}.$$

Mệnh đề 2.1 (Bauschke và Combettes [1]) *Ta có*

(i) $||P_C(x) - P_C(y)|| \leq ||x - y||$ *với mọi* x, $y \in \mathbb{R}^m$,

(ii) $\langle y - P_C(x), x - P_C(x)\rangle \leq 0$ *với mọi* $x \in \mathbb{R}^m$, $y \in C$.

Định nghĩa 2.1 (Mangasarian [13]) Hàm $f : \mathbb{R}^m \to \mathbb{R}$ được gọi là:

- lồi trên C nếu với mọi x, $y \in C$, $\lambda \in [0, 1]$, thỏa mãn rằng
$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$
- giả lồi trên C nếu với mọi x, $y \in C$, thỏa mãn rằng
$$\langle \nabla f(x), y - x\rangle \geq 0 \Rightarrow f(y) \geq f(x).$$
- tựa lồi trên C nếu với mọi x, $y \in C$, $\lambda \in [0; 1]$, thỏa mãn rằng
$$f(\lambda x + (1 - \lambda)y) \leq \max\{f(x); f(y)\}.$$

Mệnh đề 2.2 (Dennis và Schnabel [5]) *Hàm khả vi* f *là tựa lồi trên* C *khi và chỉ khi*
$$f(y) \leq f(x) \Rightarrow \langle \nabla f(x), y - x\rangle \leq 0.$$

Đáng chú ý là "f là lồi" $\Rightarrow$ "f là giả lồi" $\Rightarrow$ "f là tựa lồi" (xem Mangasarian [13]).

Mệnh đề 2.3 (Dennis và Schnabel [5]) *Giả sử rằng* ∇f *liên tục Lipschitz trên* C. *Với mọi* x, $y \in C$, *ta có*
$$|f(y) - f(x) - \langle \nabla f(x), y - x\rangle| \leq \frac{L}{2}||y - x||^2.$$

Bổ đề 2.1 (Xu [19]) *Cho* $\{a_k\}$; $\{b_k\} \subset (0; \infty)$ *là các dãy sao cho*
$$a_{k+1} \leq a_k + b_k \ \forall k \geq 0; \ \sum_{k=0}^{\infty} b_k < \infty.$$
Khi đó, tồn tại giới hạn $\lim_{k\to\infty} a_k = c \in \mathbb{R}$.

3 Các kết quả chính

Trong mục này, chúng tôi trình bày thuật toán được đề xuất và chứng minh sự hội tụ của nó. Dựa trên các mệnh đề và bổ đề đã nêu ở Mục 2, bài báo [17] đưa ra kết quả chính như sau.

Nhận xét 3.1 Nếu Thuật toán GDA dừng tại bước k, thì x^k là một điểm dừng của bài toán OP(f, C).

Thật vậy, vì $x^{k+1} = P_C(x^k - \lambda_k \nabla f(x^k))$, áp dụng Mệnh đề 2.1-(ii), ta có:
$$\langle z - x^{k+1}, x^k - \lambda_k \nabla f(x^k) - x^{k+1}\rangle \leq 0 \quad \forall z \in C. \tag{1}$$

Nếu $x^{k+1} = x^k$, ta có:
$$\langle \nabla f(x^k), z - x^k\rangle \geq 0 \quad \forall z \in C, \tag{2}$$
điều này có nghĩa là x^k là một điểm dừng của bài toán. Bên cạnh đó, nếu f là hàm giả lồi thì từ (2), ta suy ra $f(z) \geq f(x^k)$ với mọi $z \in C$, hay nói cách khác x^k là một nghiệm của bài toán OP(f, C).

Thuật toán 1 Thuật toán Thích nghi Gradient Descent (GDA)

Bước 0. Chọn $x^0 \in C,\ \lambda_0 \in (0, +\infty),\ \sigma,\ \kappa \in (0, 1)$. Đặt $k = 0$.
Bước 1. Từ x^k và λ_k, tính toán $x^{k+1} = P_C(x^k - \lambda_k \nabla f(x^k))$.
Nếu $f(x^{k+1}) \leq f(x^k) - \sigma\langle \nabla f(x^k), x^k - x^{k+1}\rangle$ **thì**
    Đặt $\lambda_{k+1} := \lambda_k$
Ngược lại
    Đặt $\lambda_{k+1} := \kappa\lambda_k$
Bước 2. Kiểm tra điều kiện dừng
Nếu $x^{k+1} = x^k$ **thì**
    DỪNG
Ngược lại
    Cập nhật $k := k + 1$ và quay lại **Bước 1**.

Bây giờ, giả sử rằng thuật toán tạo ra một dãy vô hạn. Ta sẽ chứng minh rằng dãy này hội tụ đến một nghiệm của bài toán OP(f, C).

Định lý 3.1 *Giả sử rằng dãy $\{x^k\}$ được tạo ra bởi Thuật toán GDA. Khi đó, dãy $\{f(x^k)\}$ hội tụ và mỗi điểm giới hạn (nếu có) của dãy $\{x^k\}$ là một điểm dừng của bài toán. Hơn nữa,*

- *nếu f là tựa lồi trên C, thì dãy $\{x^k\}$ hội tụ đến một điểm dừng của bài toán.*
- *nếu f là giả lồi trên C, thì dãy $\{x^k\}$ hội tụ đến một nghiệm của bài toán.*

Chứng minh. Áp dụng Mệnh đề 2.3, ta có

$$f(x^{k+1}) \leq f(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k\rangle + \frac{L}{2}\|x^{k+1} - x^k\|^2. \tag{3}$$

Ở (1), lấy $z = x^k \in C$, ta có

$$\langle \nabla f(x^k), x^{k+1} - x^k\rangle \leq -\frac{1}{\lambda_k}\|x^{k+1} - x^k\|^2. \tag{4}$$

Kết hợp (3) và (4), ta thu được

$$f(x^{k+1}) \leq f(x^k) - \sigma\langle \nabla f(x^k), x^k - x^{k+1}\rangle - \left(\frac{1-\sigma}{\lambda_k} - \frac{L}{2}\right)\|x^{k+1} - x^k\|^2. \tag{5}$$

Bây giờ ta khẳng định rằng $\{\lambda_k\}$ không tiến về 0, hay nói cách khác, kích thước cỡ bước thay đổi hữu hạn lần. Thật vậy, giả sử ngược lại rằng $\lambda_k \to 0$. Từ (5), tồn tại $k_0 \in \mathbb{N}$ thỏa mãn

$$f(x^{k+1}) \leq f(x^k) - \sigma\langle \nabla f(x^k), x^k - x^{k+1}\rangle \quad \forall k \geq k_0.$$

Theo cách xây dựng λ_k, bất đẳng thức cuối cùng cho ta thấy rằng $\lambda_k = \lambda_{k_0}$ với mọi $k \geq k_0$. Điều này là mâu thuẫn. Và do đó, tồn tại $k_1 \in \mathbb{N}$ sao cho với mọi $k \geq k_1$, ta có $\lambda_k = \lambda_{k_1}$ và

$$f(x^{k+1}) \leq f(x^k) - \sigma\langle \nabla f(x^k), x^k - x^{k+1}\rangle. \tag{6}$$

Lưu ý rằng $\langle \nabla f(x^k), x^k - x^{k+1}\rangle \geq 0$, ta suy ra rằng dãy $\{f(x^k)\}$ là hội tụ và

$$\sum_{k=0}^{\infty}\langle \nabla f(x^k), x^k - x^{k+1}\rangle < \infty; \quad \sum_{k=0}^{\infty}\|x^{k+1} - x^k\|^2 < \infty. \tag{7}$$

Từ (1), với mọi $z \in C$, ta có

$$\begin{aligned}\|x^{k+1} - z\|^2 &= \|x^k - z\|^2 - \|x^{k+1} - x^k\|^2 + 2\langle x^{k+1} - x^k, x^{k+1} - z\rangle \\ &\leq \|x^k - z\|^2 - \|x^{k+1} - x^k\|^2 + 2\lambda_k\langle \nabla f(x^k), z - x^{k+1}\rangle. \end{aligned} \tag{8}$$

Gọi $\bar{x}$ là một điểm giới hạn của $\{x^k\}$. Tồn tại một dãy con $\{x^{k_i}\} \subset \{x^k\}$ sao cho $\lim_{i\to\infty} x^{k_i} = \bar{x}$. Trong (8), đặt $k = k_i$ và lấy giới hạn khi $i \to \infty$. Lưu ý rằng $\|x^k - x^{k+1}\| \to 0$, ∇f là liên tục, ta có

$$\langle \nabla f(\bar{x}), z - \bar{x}\rangle \geq 0 \quad \forall z \in C,$$

điều này có nghĩa là $\bar{x}$ là một điểm dừng của bài toán.

Bây giờ, giả sử rằng f là tựa lồi trên C. Ký hiệu

$$U := \{x \in C : f(x) \leq f(x^k) \quad \forall k \geq 0\}.$$

Lưu ý rằng U chứa tập nghiệm của OP(f, C), nên nó không rỗng. Lấy $\hat{x} \in U$. Vì $f(x^k) \geq f(\hat{x})$ với mọi $k \geq 0$, ta có

$$\langle \nabla f(x^k), \hat{x} - x^k\rangle \leq 0 \quad \forall k \geq 0. \tag{9}$$

Kết hợp (8) và (9), ta có

$$\|x^{k+1} - \hat{x}\|^2 \leq \|x^k - \hat{x}\|^2 - \|x^{k+1} - x^k\|^2 + 2\lambda_k\langle \nabla f(x^k), x^k - x^{k+1}\rangle. \tag{10}$$

Áp dụng Bổ đề 2.1 với $a_k = \|x^k - \hat{x}\|^2$, $b_k = 2\lambda_k\langle \nabla f(x^k), x^k - x^{k+1}\rangle$, ta suy ra rằng dãy $\{\|x^k - \hat{x}\|\}$ là hội tụ với mọi $\hat{x} \in U$. Vì dãy $\{x^k\}$ là bị chặn, tồn tại một dãy con $\{x^{k_i}\} \subset \{x^k\}$ sao cho $\lim_{i\to\infty} x^{k_i} = \bar{x} \in C$. Từ (6), ta biết rằng dãy $\{f(x^k)\}$ là không tăng và hội tụ. Từ đó ta có $\lim_{k\to\infty} f(x^k) = f(\bar{x})$ và $f(\bar{x}) \leq f(x^k)$ với mọi $k \geq 0$. Điều này có nghĩa là $\bar{x} \in U$ và dãy $\{\|x^k - \bar{x}\|\}$ là hội tụ.

Do đó,

$$\lim_{k\to\infty} \|x^k - \bar{x}\| = \lim_{i\to\infty} \|x^{k_i} - \bar{x}\| = 0.$$

Lưu ý rằng mỗi điểm giới hạn của $\{x^k\}$ là một điểm dừng của bài toán. Khi đó, toàn bộ dãy $\{x^k\}$ hội tụ về $\bar{x}$ – một điểm dừng của bài toán. Hơn nữa, khi f là giả lồi, điểm dừng này trở thành một nghiệm của OP(f, C). □

Nhận xét 3.2 Trong Thuật toán GDA, ta có thể chọn $\lambda_0 = \lambda$, với hằng số $\lambda \leq 2(1-\sigma)/L$. Khi đó, ta có $(1-\sigma)/\lambda_0 - L/2 \geq 0$. Kết hợp với (5), ta thấy rằng điều kiện $f(x^{k+1}) \leq f(x^k) - \sigma\langle \nabla f(x^k), x^k - x^{k+1}\rangle$ được thỏa mãn và cỡ bước $\lambda_k = \lambda$ cho mọi bước k. Do đó, Thuật toán GDA vẫn có thể áp dụng cho cỡ bước là hằng số $\lambda \leq 2(1-\sigma)/L$. Đối với bất kỳ $\lambda \in (0, 2/L)$, tồn tại $\sigma \in (0,1)$ sao cho $\lambda \leq 2(1-\sigma)/L$. Kết quả là, nếu giá trị của hằng số Lipschitz L đã được biết trước, ta có thể chọn cỡ bước là hằng số $\lambda \in (0, 2/L)$ như trong thuật toán gradient descent (GD) để giải các bài toán quy hoạch lồi. Thuật toán GD này đã được đề xuất trong các công trình trước đây. Vì nó là một trường hợp đặc biệt của Thuật toán GDA, sự hội tụ của nó được đảm bảo như các khẳng định trong Định lý 3.1.

Lưu ý rằng tất cả các khẳng định của Định lý 3.1 vẫn đúng cho dãy $\{x^k\}$ được tạo ra bởi Thuật toán GD. Bây giờ, ta ước lượng tốc độ hội tụ của Thuật toán GDA trong việc giải các bài toán tối ưu hóa không ràng buộc.

Thuật toán 2 Thuật toán Gradient Descent (GD)

1: **Bước 0.** Chọn $x^0 \in C,\ \lambda \in (0, 2/L)$. Đặt $k = 0$.
2: **Bước 1.** Từ x^k, tính toán $x^{k+1} = P_C(x^k - \lambda \nabla f(x^k))$
3: **Bước 2.** Kiểm tra điều kiện dừng
4: **Nếu** $x^{k+1} = x^k$ **thì**
5: DỪNG
6: **Ngược lại**
7: Cập nhật $k := k + 1$ và quay lại **Bước 1**.

Hệ quả 3.1 *Giả sử rằng f là lồi, $C = \mathbb{R}^m$ và $\{x^k\}$ là dãy được tạo ra bởi Thuật toán GDA. Khi đó,*

$$f(x^k) - f(x^*) = O\left(\frac{1}{k}\right)$$

trong đó x^* là một nghiệm của bài toán.

Chứng minh. Gọi x^* là một nghiệm của bài toán. Ký hiệu $\Delta_k := f(x^k) - f(x^*)$. Từ (6), lưu ý rằng $x^k - x^{k+1} = \lambda_k \nabla f(x^k)$, ta có

$$\Delta_{k+1} \leq \Delta_k - \sigma \lambda_{k_1} \|\nabla f(x^k)\|^2 \quad \forall k \geq k_1. \tag{11}$$

Mặt khác, vì dãy $\{x^k\}$ bị chặn và f là lồi, ta có

$$\begin{aligned} 0 \leq \Delta_k &\leq \langle \nabla f(x^k), x^k - x^* \rangle \\ &\leq M \|\nabla f(x^k)\|, \end{aligned} \tag{12}$$

trong đó $M := \sup\{\|x^k - x^*\| : k \geq k_1\} < \infty$. Từ (11) và (12), ta có

$$\Delta_{k+1} \leq \Delta_k - Q\Delta_k^2 \quad \forall k \geq k_1, \tag{13}$$

trong đó $Q := \frac{\sigma \lambda_{k_1}}{M^2}$. Lưu ý rằng $\Delta_{k+1} \leq \Delta_k$, từ (13), ta thu được

$$\frac{1}{\Delta_{k+1}} \geq \frac{1}{\Delta_k} + Q \geq \cdots \geq \frac{1}{\Delta_{k_1}} + (k - k_1)Q,$$

từ đó suy ra

$$f(x^k) - f(x^*) = O\left(\frac{1}{k}\right).$$

□

Để kết thúc phần này, bài báo [17] trình bày một biến thể ngẫu nhiên của Thuật toán GDA để áp dụng trong học sâu quy mô lớn. Xét bài toán:

$$\min_x \mathbb{E}[f_\xi(x)],$$

trong đó ξ là tham số ngẫu nhiên và hàm f_ξ là L-smooth (tức là gradient của nó liên tục Lipschitz). Ta đang tạo ra một gradient ngẫu nhiên $\nabla f_{\xi^k}(x^k)$ bằng cách lấy mẫu ξ^k tại mỗi lần lặp k. Biến thể ngẫu nhiên của phương pháp gradient descent, đặc biệt là trong bối cảnh học sâu quy mô lớn, đóng vai trò quan trọng trong việc tối ưu hóa các mô hình phức tạp một cách hiệu quả. Khi ta xem xét bài toán tối ưu hóa có dạng:

$$\min_x \mathbb{E}[f_\xi(x)],$$

trong đó x đại diện cho các tham số của mô hình (như các trọng số trong mạng nơ-ron), ξ là một tham số ngẫu nhiên, và $f_\xi(x)$ là một hàm L-smooth, chúng tôi đang đối phó với một kịch bản nơi hàm mục tiêu được định nghĩa là giá trị kỳ vọng của một số hàm ngẫu nhiên $f_\xi(x)$ nào đó. Khuôn khổ này là điển hình trong học máy, nơi $f_\xi(x)$ thường đại diện cho hàm mất mát (loss function) được tính toán trên một tập con (lô/batch) của dữ liệu huấn luyện, và ξ đại diện cho tính ngẫu nhiên trong việc lựa chọn tập con này.

Dưới đây là mô tả chi tiết của thuật toán SGDA. Các tác giả dành lại các kết quả lý thuyết của Thuật toán SGDA cho các nghiên cứu sau này.

Thuật toán 3 Thuật toán Thích nghi Gradient Descent Ngẫu nhiên (SGDA)

Bước 0. Chọn $x^0 \in C,\ \lambda_0 \in (0, +\infty),\ \sigma,\ \kappa \in (0, 1)$. Đặt $k = 0$.
Bước 1. Lấy mẫu ξ^k và tính $x^{k+1} = P_C(x^k - \lambda_k \nabla f_{\xi^k}(x^k))$.
Nếu $f_{\xi^k}(x^{k+1}) \leq f_{\xi^k}(x^k) - \sigma\langle \nabla f_{\xi^k}(x^k), x^k - x^{k+1}\rangle$ **thì**
    Đặt $\lambda_{k+1} := \lambda_k$
Ngược lại
    Đặt $\lambda_{k+1} := \kappa\lambda_k$
Bước 2. Kiểm tra điều kiện dừng
Nếu $x^{k+1} = x^k$ **thì**
    DỪNG
Ngược lại
    Cập nhật $k := k + 1$ và quay lại **Bước 1**.

4 Kết quả thực nghiệm

Trong phần này, các tác giả sử dụng hai ví dụ hiện có và hai ví dụ quy mô lớn với các kích thước thay đổi để xác nhận hiệu quả của phương pháp đề xuất. Các thí nghiệm ứng dụng trong học máy sẽ được thực hiện ở phần tiếp theo. Nhóm chúng tôi đã triển khai lại các ví dụ, mã nguồn có sẵn tại: https://github.com/tuasananh/NMPPTU-Team1-SAAQP.git.

Tất cả các thử nghiệm được thực hiện bằng Python trên laptop với CPU Intel Core i9-13900HX (24 lõi với xung nhịp cơ bản 2.2GHz) và 32GB RAM. Tiêu chí dừng trong các trường hợp sau là "số vòng lặp $\leq$ #Iter" trong đó #Iter là số vòng lặp tối đa. Ký hiệu x^* là điểm giới hạn của dãy lặp $\{x_k\}$ và Time là thời gian CPU của thuật toán GDA khi sử dụng tiêu chí dừng.

Các hàm mục tiêu và ràng buộc không lồi được minh họa trong Ví dụ 1 và 2 lấy từ [11]. Ngoài ra, Ví dụ 2 phức tạp hơn Ví dụ 1 về dạng hàm. Việc thực hiện Ví dụ 3 với hàm mục tiêu lồi và số chiều n thay đổi cho phép đánh giá phương pháp đề xuất so với Thuật toán GD. Ví dụ 4 được thực hiện với hàm mục tiêu giả lồi và một số giá trị n để đánh giá phương pháp được các tác giả đề xuất so với Thuật toán RNN.

Để so sánh với phương pháp neurodynamic trong [11], bài báo xét bài toán $\mathrm{OP}(f, C)$ với tập ràng buộc được xác định bởi $C = \{x \in \mathbb{R}^n \mid g(x) \leq 0,\ Ax = b\}$, trong đó $g(x) := (g_1(x), g_2(x), \ldots, g_m(x))^\top$ và $g_i : \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, m$ là các hàm tựa lồi khả vi, ma trận $A \in \mathbb{R}^{p\times n}$ và $b = (b_1, b_2, \ldots, b_p)^\top \in \mathbb{R}^p$. Trong [11], các

tác giả đã giới thiệu hàm

$$\Psi(s) = \begin{cases} 1, & s > 0, \\ [0,1], & s = 0, \\ 0, & s < 0. \end{cases}$$

và

$$P(x) = \sum_{i=1}^{m} \max\{0, g_i(x)\}.$$

Thuật toán neurodynamic được thiết lập trong [11], sử dụng các mô hình mạng nơ-ron hồi quy (RNN) để giải bài toán OP(f, C) dưới dạng bao hàm vi phân như sau:

$$\frac{d}{dt}x(t) \in -c(x(t))\nabla f(x(t)) - \partial P(x(t)) - \partial\|Ax(t) - b\|_1 \quad (RNN)$$

trong đó hệ số điều chỉnh $c(x(t))$ là

$$c(x(t)) = \left\{ \prod_{i=1}^{m+p} c_i(t) \;\middle|\; c_i(t) \in 1 - \Psi(J_i(x(t))),\ i = 1, 2, \ldots, m+p \right\}$$

với

$$J(x) = \big(g_1(x), \ldots, g_m(x), |A_1x - b_1|, \ldots, |A_px - b_p|\big)^\top,$$

hạng tử subgradient của $P(x)$ là

$$\partial P(x) = \begin{cases} 0, & x \in \text{int}(X) \\ \sum_{i\in I_0(x)} [0,1]\nabla g_i(x), & x \in \text{bd}(X) \\ \sum_{i\in I_+(x)} \nabla g_i(x) + \sum_{i\in I_0(x)} [0,1]\nabla g_i(x), & x \notin X, \end{cases}$$

với

$$X = \{x : g_i(x) \le 0,\ i = 1, 2, \ldots, m\},$$
$$I_+(x) = \{i \in \{1, 2, \ldots, m\} : g_i(x) > 0\},$$
$$I_0(x) = \{i \in \{1, 2, \ldots, m\} : g_i(x) = 0\},$$

và hạng tử subgradient của $\|Ax - b\|_1$ là

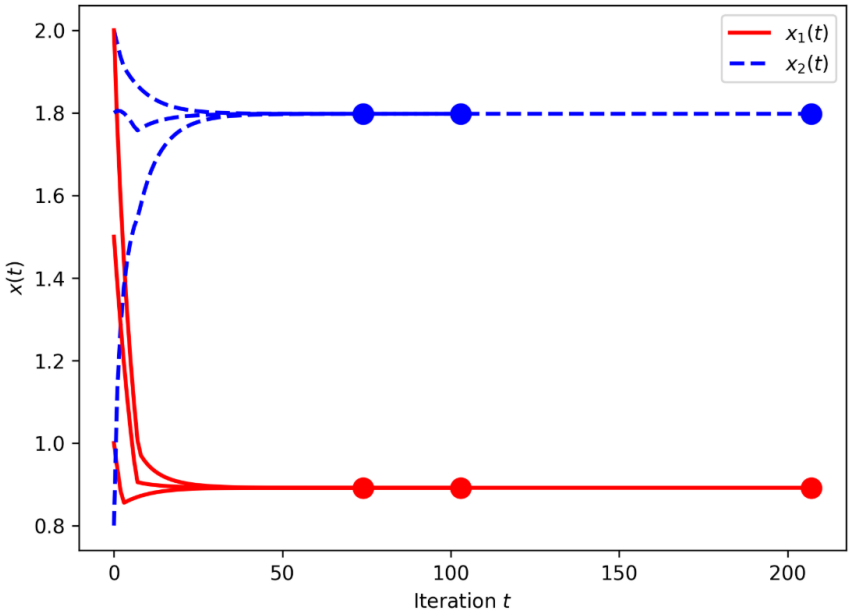
$$\partial\|Ax - b\|_1 = \sum_{i=1}^{p} \big(2\Psi(A_ix - b_i) - 1\big)A_i^\top,$$

với $A_i \in \mathbb{R}^{1\times n}$ $(i = 1, 2, \ldots, p)$ là các vector dòng của ma trận A.

Ví dụ 1 Trước hết, xét một bài toán phi lồi đơn giản OP(f, C):

$$\begin{aligned} \min \quad & f(x) = \frac{x_1^2 + x_2^2 + 3}{1 + 2x_1 + 8x_2} \\ \text{v.đ.k} \quad & x \in C, \end{aligned}$$

trong đó $C = \{x = (x_1, x_2)^\top \in \mathbb{R}^2 \mid g_1(x) = -x_1^2 - 2x_1x_2 \le -4;\ x_1, x_2 \ge 0\}$. Trong bài toán này, hàm mục tiêu f là giả lồi trên tập khả thi lồi (Ví dụ 5.2 trong [11]).



Hình 1 Kết quả tính toán cho Ví dụ 1.

Hình 1 minh họa các nghiệm tạm thời của phương pháp đề xuất với các nghiệm ban đầu khác nhau. Nó cho thấy các kết quả hội tụ đến nghiệm tối ưu $x^* = (0.8922,\ 1.7957)$ của bài toán đã cho. Giá trị hàm mục tiêu do Thuật toán GDA tạo ra là 0.4094, tốt hơn giá trị tối ưu 0.4101 của mạng nơ-ron trong [11].

Ví dụ 2 Xét bài toán tối ưu giả lồi không trơn với các ràng buộc bất đẳng thức không lồi sau (Ví dụ 5.1 trong [11]):

$$\begin{aligned} \text{minimize} \quad & f(x) = \frac{e^{|x_2-3|} - 30}{x_1^2 + x_3^2 + 2x_4^2 + 4} \\ \text{subject to} \quad & g_1(x) = (x_1 + x_3)^3 + 2x_4^2 \le 10, \\ & g_2(x) = (x_2 - 1)^2 \le 1, \\ & 2x_1 + 4x_2 + x_3 = -1, \end{aligned}$$

trong đó $x = (x_1, x_2, x_3, x_4)^\top \in \mathbb{R}^4$. Hàm mục tiêu $f(x)$ là không trơn giả lồi trên miền khả thi C, và ràng buộc bất đẳng thức g_1 là liên tục giả lồi trên C, nhưng không giả lồi (xem [11]). Dễ thấy rằng $x_2 \neq 3$ với mọi $x \in C$. Do đó, vector gradient của $|x_2-3|$ là $(x_2-3)/|x_2-3|$ với mọi $x \in C$. Từ đó, chúng ta có thể thiết lập vector gradient của $f(x)$ tại điểm $x \in C$ dùng trong thuật toán. Hình 2 cho thấy Thuật toán GDA hội tụ đến nghiệm tối ưu $x^* = (-1.0649,\ 0.4160,\ -0.5343,\ 0.0002)^\top$ với giá trị tối ưu -3.0908, tốt hơn giá trị tối ưu -3.0849 của mô hình mạng nơ-ron trong [11].

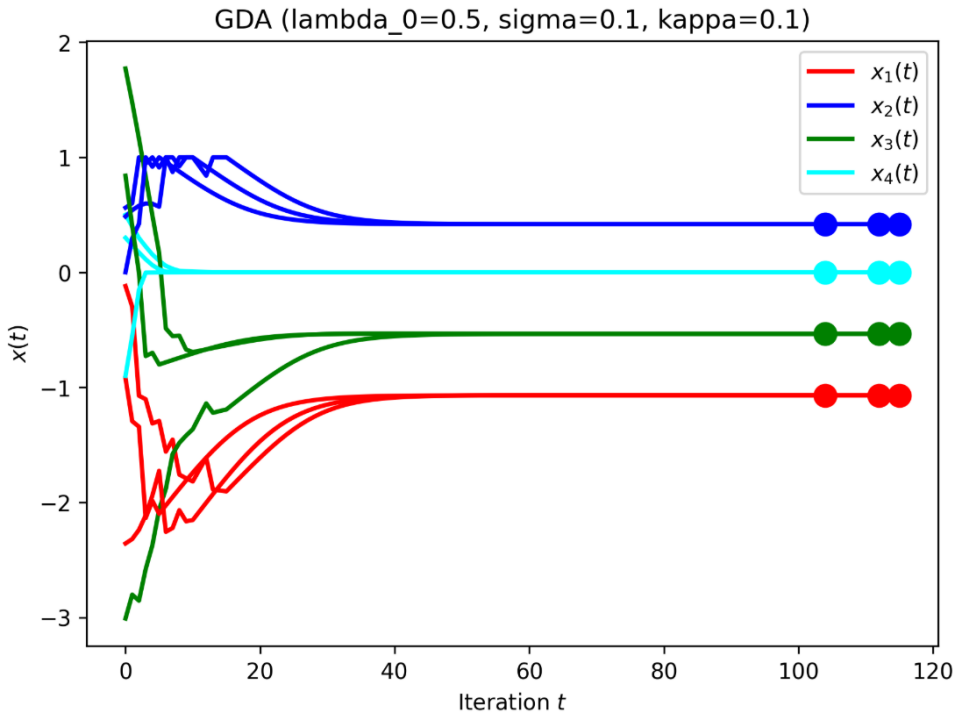
Ví dụ 3 Đặt $e := (1, \ldots, n) \in \mathbb{R}^n$ là vector, $\alpha > 0$ và $\beta > 0$ là các hằng số thỏa mãn điều kiện tham số $2\alpha > 3\beta^{3/2}\sqrt{n}$. Xét bài toán $\mathrm{OP}(f, C)$ (Ví dụ 4.5 trong [6]) với hàm tương ứng

$$f(x) := a^\top x + \alpha x^\top x + \frac{\beta}{\sqrt{1 + \beta x^\top x}} e^\top x,$$

với $a \in \mathbb{R}^n$ là lồi và ràng buộc không lồi được cho bởi

$$C := \{x \in \mathbb{R}^n_+ : 1 \le x_1 \ldots x_n\}.$$

Ví dụ này được thực hiện để so sánh Thuật toán GDA với thuật toán gradient descent gốc (GD). Thí nghiệm được thực hiện với hằng số được chọn ngẫu nhiên



Hình 2 Kết quả tính toán cho Ví dụ 2.

$\beta = 0.741271$, $\alpha = 3\beta^{3/2}\sqrt{n} + 1$ thỏa mãn điều kiện tham số và hệ số Lipschitz $L = (4\beta^{3/2}\sqrt{n} + 3\alpha)$ được gợi ý trong [6]. Cỡ bước của Thuật toán GD là $\lambda = 1/L$, và cỡ bước ban đầu của Thuật toán GDA là $\lambda_0 = 5/L$.

Hai kết quả thực nghiệm dưới đây cho thấy rằng với cỡ bước được chọn dựa theo hằng số Lipschitz L, cả hai thuật toán đều hội tụ rất nhanh (không quá 20 vòng lặp). Nguyên nhân cụ thể dẫn đến tốc độ hội tụ này vẫn là một vấn đề mở cần được nghiên cứu thêm.

Kết quả chi tiết với hệ số nhân `gda_multiplier` $= 5.0$ (cỡ bước ban đầu của GDA gấp 5 lần so với GD) được trình bày tại Bảng 1.

Bảng 1 Kết quả cho Ví dụ 3 với cỡ bước tự thích nghi (gda_multiplier = 5.0)

n	Thuật toán GDA (đề xuất)			Thuật toán GD		
	$f(x^*)$	#Iter	Time (ms)	$f(x^*)$	#Iter	Time (ms)
10	80.0806	18	5.19	80.0806	19	5.34
20	219.5903	17	4.45	219.5903	19	4.82
50	852.4965	17	5.09	852.4965	19	3.76
100	2394.3380	17	2.68	2394.3380	20	9.16
200	6732.2122	17	8.96	6732.2122	20	8.92
500	26429.0556	17	11.01	26429.0556	20	3.30
1000	74531.2658	17	4.31	74531.2658	20	8.61
2000	210371.4228	17	6.40	210371.4228	20	7.23
3000	386136.1503	17	7.88	386136.1503	20	7.29
10000	2345979.1282	17	20.03	2345979.1282	20	19.42

Với `gda_multiplier` $= 2$, nghĩa là cỡ bước ban đầu của GDA gấp đôi so với GD, thuật toán GDA thể hiện hiệu năng tốt hơn hẳn (xem Bảng 2). Điều này có thể được giải thích như sau: Thuật toán hội tụ nhanh do việc chọn cỡ bước dựa theo L đã là một lựa chọn rất tốt. Nếu tăng cỡ bước ban đầu lên cao, thuật toán sẽ tốn một vài bước lặp để điều chỉnh cỡ bước trở về mức lý tưởng. Việc chọn cỡ bước gấp đôi GD giúp đảm bảo cân bằng giữa tốc độ hội tụ và tính an toàn.

Bảng 2 Kết quả cho Ví dụ 3 với cỡ bước tự thích nghi (gda_multiplier = 2.0)

n	Thuật toán GDA (đề xuất)			Thuật toán GD		
	$f(x^*)$	#Iter	Time (ms)	$f(x^*)$	#Iter	Time (ms)
10	80.0806	8	5.17	80.0806	19	10.43
20	219.5903	8	3.69	219.5903	19	5.55
50	852.4965	8	5.49	852.4965	19	12.07
100	2394.3380	8	2.47	2394.3380	20	9.48
200	6732.2122	8	0.00	6732.2122	20	4.95
500	26429.0556	9	5.89	26429.0556	20	4.68
1000	74531.2658	9	2.90	74531.2658	20	5.96
2000	210371.4228	9	1.00	210371.4228	20	0.00
3000	386136.1503	9	12.05	386136.1503	20	8.28
10000	2345979.1282	9	8.66	2345979.1282	20	19.54

Trong trường hợp cỡ bước cố định, kết quả nhận được tương đồng với các công bố trong bài báo [17]. Vì cỡ bước chưa phải là lý tưởng, thuật toán GD tiêu tốn nhiều thời gian để tìm được nghiệm tối ưu.

Tương tự nhận xét ở trên, với cỡ bước ban đầu quá lớn (gấp 5 lần GD), thuật toán GDA mất nhiều thời gian để tìm lại cỡ bước lý tưởng. Do đó, với $n \leq 2000$, thuật toán thể hiện kém hơn GD (Bảng 3). Tuy nhiên, với n lớn hơn, thuật toán đã giảm đáng kể thời gian tìm nghiệm do số vòng lặp để tìm cỡ bước lý tưởng chỉ chiếm một phần nhỏ trong tổng số vòng lặp.

Bảng 3 Kết quả cho Ví dụ 3 với cỡ bước cố định (gda_multiplier = 5.0, step_size = 0.1)

n	Thuật toán GDA (đề xuất)			Thuật toán GD		
	$f(x^*)$	#Iter	Time (ms)	$f(x^*)$	#Iter	Time (ms)
10	80.0806	46	15.07	80.0806	8	0.00
20	219.5903	58	17.90	219.5903	13	2.62
50	852.4965	90	21.99	852.4965	21	5.73
100	2394.3380	111	30.87	2394.3380	29	8.03
200	6732.2122	100	27.75	6732.2122	41	11.73
500	26429.0556	130	40.58	26429.0556	66	20.92
1000	74531.2658	112	44.95	74531.2658	92	34.48
2000	210371.4228	138	61.49	210371.4228	129	57.46
3000	386136.1503	92	50.98	386136.1503	161	94.66
10000	2345979.1282	101	137.50	2345979.1282	293	426.79

Thật vậy, khi GDA xuất phát với cỡ bước gấp 2 lần GD, thuật toán thể hiện tốt hơn rõ rệt, đặc biệt với $n \geq 2000$. Thuật toán GDA chỉ thực sự mạnh hơn đáng kể so với GD khi n đủ lớn trong trường hợp cỡ bước cố định (Bảng 4).

Bảng 4 Kết quả cho Ví dụ 3 với cỡ bước cố định (gda_multiplier = 2.0, step_size = 0.1)

n	Thuật toán GDA (đề xuất)			Thuật toán GD		
	$f(x^*)$	#Iter	Time (ms)	$f(x^*)$	#Iter	Time (ms)
10	80.0806	19	8.71	80.0806	8	0.00
20	219.5903	26	8.13	219.5903	13	3.47
50	852.4965	41	11.98	852.4965	21	6.46
100	2394.3380	55	9.68	2394.3380	29	11.73
200	6732.2122	78	19.77	6732.2122	41	10.18
500	26429.0556	109	31.90	26429.0556	66	22.56
1000	74531.2658	103	32.23	74531.2658	92	36.17
2000	210371.4228	114	51.38	210371.4228	129	65.05
3000	386136.1503	130	84.40	386136.1503	161	95.98
10000	2345979.1282	125	172.49	2345979.1282	293	400.85

Ví dụ 4 Để so sánh Thuật toán GDA với Thuật toán RNN trong [11], xét bài toán OP(f, C) với hàm mục tiêu

$$f(x) = -\exp\left(-\sum_{i=1}^{n} \frac{x_i^2}{\varrho_i^2}\right)$$

là giả lồi trên tập ràng buộc lồi

$$C := \{Ax = b,\ g(x) \leq 0\},$$

trong đó $x \in \mathbb{R}^n$, vector tham số $\varrho = (\varrho_1, \varrho_2, \ldots, \varrho_n)^\top$ với $\varrho_i > 0$, ma trận $A = (a_1, a_2, \ldots, a_n) \in \mathbb{R}^{1\times n}$ với $a_i = 1$ khi $1 \leq i \leq n/2$ và $a_i = 3$ khi $n/2 < i \leq n$, và số $b = 16$. Các ràng buộc bất đẳng thức là

$$g_i(x) = x^2_{10\cdot(i-1)+1} + x^2_{10\cdot(i-1)+2} + \cdots + x^2_{10\cdot(i-1)+10} - 20,$$

với $i = 1, 2, \ldots, n/10$. Bảng 5 trình bày kết quả tính toán của Thuật toán GDA và RNN. Lưu ý rằng hàm $-\ln(-z)$ là đơn điệu tăng với $z \in \mathbb{R}$, $z < 0$. Do đó, để so sánh giá trị tối ưu xấp xỉ qua các vòng lặp, các tác giả tính $-\ln(-f(x^*))$ thay vì $f(x^*)$, với nghiệm tối ưu xấp xỉ x^*. Với mỗi n, bài báo [17] giải bài toán OP(f, C) để tìm giá trị $-\ln(-f(x^*))$, số vòng lặp (#Iter) để dừng thuật toán, và thời gian tính toán (Time) tính bằng giây. Kết quả tính toán cho thấy thuật toán đề xuất vượt trội hơn Thuật toán RNN trong các kịch bản thử nghiệm cả về giá trị tối ưu và thời gian tính toán, đặc biệt khi chiều lớn.

Bảng 5 Kết quả tính toán cho Ví dụ 4

n	Thuật toán GDA (đề xuất)			Thuật toán RNN		
	$-\ln(-f(x^*))$	#Iter	Time	$-\ln(-f(x^*))$	#Iter	Time
10	5.1200	10	0.3	5.1506	1000	256
20	2.5600	10	0.8	2.5673	1000	503
50	1.0240	10	2	1.0299	1000	832
100	0.5125	10	7	13.7067	1000	1420
300	15.7154	10	84	39.3080	1000	3292
400	20.9834	10	163	57.6837	1000	4426
600	29.0228	10	371	83.6265	1000	6788

5 Ứng dụng trong học máy

Phương pháp được đề xuất, giống như thuật toán GD, có nhiều ứng dụng trong học máy. Trong báo cáo này, nhóm chúng tôi trình bày và tái triển khai ba thực nghiệm ứng với ba ứng dụng tiêu biểu được trình bày trong bài báo [17], bao gồm lựa chọn đặc trưng có giám sát (*supervised feature selection*), hồi quy (*regression*) và phân loại (*classification*), để chứng minh độ chính xác và hiệu quả tính toán so với các phương pháp thay thế khác.

Đầu tiên, bài toán lựa chọn đặc trưng có thể được mô hình hóa như một bài toán cực tiểu hóa một hàm phân thức giả lồi trên một tập lồi, đây là một lớp con của Bài toán OP(f, C). Bài toán này được sử dụng để so sánh phương pháp tiếp cận được đề xuất với các phương pháp thần kinh động (*neurodynamic approaches*).

Thứ hai, vì bài toán hồi quy logistic đa biến là một bài toán quy hoạch lồi, nên thuật toán GDA và các biến thể có sẵn của thuật toán GD có thể được sử dụng để giải nó.

Cuối cùng, một mô hình mạng nơ-ron cho bài toán phân loại ảnh cũng giống như một bài toán quy hoạch với hàm mục tiêu không lồi cũng không tựa lồi. Để huấn luyện mô hình này, các tác giả sử dụng biến thể ngẫu nhiên của phương pháp GDA (Thuật toán SGDA) như một kỹ thuật heuristic. Mặc dù sự hội tụ của thuật toán không thể được đảm bảo như trong các trường hợp hàm mục tiêu giả lồi và tựa lồi, nghiên cứu lý thuyết đã chỉ ra rằng nếu dãy điểm có một điểm giới hạn, nó sẽ hội tụ về một điểm dừng của bài toán (xem Định lý 3.1). Các thực nghiệm tính toán chỉ ra rằng phương pháp được đề xuất vượt trội hơn so với các phương pháp thần kinh động và gradient descent hiện có. Cuối cùng, một mô hình mạng nơ-ron cho bài toán phân loại ảnh cũng giống như một bài toán quy hoạch với hàm mục tiêu không lồi cũng không tựa lồi. Để huấn luyện mô hình này, các tác giả sử dụng biến thể ngẫu nhiên của phương pháp GDA (Thuật toán SGDA) như một kỹ thuật heuristic. Mặc dù sự hội tụ của thuật toán không thể được đảm bảo như trong các trường hợp hàm mục tiêu giả lồi và tựa lồi, nghiên cứu lý thuyết đã chỉ ra rằng nếu dãy điểm có một điểm giới hạn, nó sẽ hội tụ về một điểm dừng của bài toán (xem Định lý 3.1). Các thực nghiệm tính toán chỉ ra rằng phương pháp được đề xuất vượt trội hơn so với các phương pháp thần kinh động và gradient descent hiện có.

5.1 Chọn đặc trưng có giám sát

Bài toán chọn đặc trưng được thực hiện trên tập dữ liệu với tập gồm p đặc trưng $\mathcal{F} = \{F_1, \ldots, F_p\}$ và tập gồm n mẫu $\{(x_i, y_i) \mid i = 1, \ldots, n\}$, trong đó $x_i = (x_{i1}, \ldots, x_{ip})^T$ là vectơ đặc trưng p chiều của mẫu thứ i và $y_i \in \{1, \ldots, m\}$ đại diện cho các nhãn tương ứng chỉ ra các lớp hoặc giá trị mục tiêu. Trong [18], một tập con tối ưu gồm k đặc trưng $\{F_1, \ldots, F_k\} \subseteq \mathcal{F}$ được chọn với độ dư thừa thấp nhất và độ liên quan cao nhất đến lớp mục tiêu y. Độ dư thừa của đặc trưng được đặc trưng bởi một ma trận bán xác định dương Q. Khi đó, mục tiêu đầu tiên là cực tiểu hóa hàm bậc hai lồi $w^T Q w$. Độ liên quan của đặc trưng được đo lường bởi $\rho^T w$, trong đó $\rho = (\rho_1, \ldots, \rho_p)^T$ là một vectơ tham số liên quan. Do đó, mục tiêu thứ hai là cực đại hóa hàm tuyến tính $\rho^T w$. Kết hợp hai mục tiêu này, ta suy ra bài toán tương đương như sau:

$$\begin{aligned} \min \quad & \frac{w^T Q w}{\rho^T w} \\ \text{v.đ.k} \quad & e^T w = 1 \\ & w \geq 0, \end{aligned} \tag{14}$$

trong đó $w = (w_1, \ldots, w_p)^T$ là vectơ điểm số đặc trưng cần xác định. Vì hàm mục tiêu của bài toán (14) có dạng phân thức, trong đó tử là một hàm lồi và mẫu là một hàm tuyến tính dương, nên nó giả lồi trên tập ràng buộc. Do đó, chúng ta có thể giải bài toán (14) bằng Thuật toán GDA.

Trong thực nghiệm, bài báo [17] triển khai các thuật toán với tập dữ liệu Parkinsons, bao gồm 23 đặc trưng và 197 mẫu, được tải xuống tại https://archive.ics.uci.edu/ml/datasets/parkinsons. Ma trận hệ số tương đồng Q được xác định bởi $Q = \delta I_p + S$ [18], trong đó $S = (s_{ij})$ là ma trận $p \times p$ với:

$$s_{ij} = \max\left\{0, \frac{I(F_i; F_j; y)}{H(F_i) + H(F_j)}\right\},$$

entropy thông tin của một vectơ biến ngẫu nhiên $\hat{X}$ là:

$$H(\hat{X}) = -\sum_{\hat{x} \in \hat{X}} p(\hat{x}) \log p(\hat{x}),$$

đa thông tin (multi-information) của ba vectơ ngẫu nhiên $\hat{X}, \hat{Y}, \hat{Z}$ là $I(\hat{X}; \hat{Y}; \hat{Z}) = I(\hat{X}; \hat{Y}) - I(\hat{X}; \hat{Y} \mid \hat{Z})$ với thông tin tương hỗ của hai vectơ ngẫu nhiên $\hat{X}, \hat{Y}$ được định nghĩa bởi:

$$I(\hat{X}; \hat{Y}) = \sum_{\hat{x} \in \hat{X}} \sum_{\hat{y} \in \hat{Y}} p(\hat{x}, \hat{y}) \log \frac{p(\hat{x}, \hat{y})}{p(\hat{x}) p(\hat{y})}$$

và thông tin tương hỗ có điều kiện giữa $\hat{X}, \hat{Y}$ và $\hat{Z}$ được định nghĩa bởi:

$$I(\hat{X}; \hat{Y} \mid \hat{Z}) = \sum_{\hat{x} \in \hat{X}} \sum_{\hat{y} \in \hat{Y}} \sum_{\hat{z} \in \hat{Z}} p(\hat{x}, \hat{y}, \hat{z}) \log \frac{p(\hat{x}, \hat{y} \mid \hat{z})}{p(\hat{x} \mid \hat{z}) p(\hat{y} \mid \hat{z})}.$$

Vectơ độ liên quan đặc trưng $\rho = (\rho_1, \ldots, \rho_p)^T$ được xác định bởi điểm Fisher:

$$\rho(F_i) = \frac{\sum_{j=1}^{K} n_j (\mu_{ij} - \mu_i)^2}{\sum_{j=1}^{K} n_j \sigma_{ij}^2},$$

trong đó n_j biểu thị số lượng mẫu trong lớp j, μ_{ij} biểu thị giá trị trung bình của đặc trưng F_i cho các mẫu trong lớp j, μ_i là giá trị trung bình của đặc trưng F_i, và σ_{ij}^2 biểu thị giá trị phương sai của đặc trưng F_i cho các mẫu trong lớp j. Bảng 6 so sánh hiệu năng của thuật toán GDA với Scipy.

Bảng 6 So sánh hiệu năng giữa thuật toán GDA và thư viện Scipy

Seed	Thuật toán GDA (đề xuất)			Scipy		
	$f(x^*)$	#Iter	Time (ms)	$f(x^*)$	#Iter	Time (ms)
1	0.152478	32	0.0186	0.152478	19	0.0062
11	0.153480	51	0.0967	0.153480	15	0.0302
21	0.153834	31	0.0160	0.153834	18	0.0062
31	0.153501	55	0.0299	0.153501	16	0.0055
41	0.154281	50	0.0260	0.154281	13	0.0040
51	0.154185	60	0.0312	0.154185	15	0.0059
61	0.154878	57	0.0343	0.154878	16	0.0062
71	0.153465	33	0.0217	0.153465	16	0.0056
81	0.153517	59	0.0318	0.153517	13	0.0035
91	0.153269	52	0.0302	0.153269	15	0.0050

5.2 Hồi quy logistic đa biến

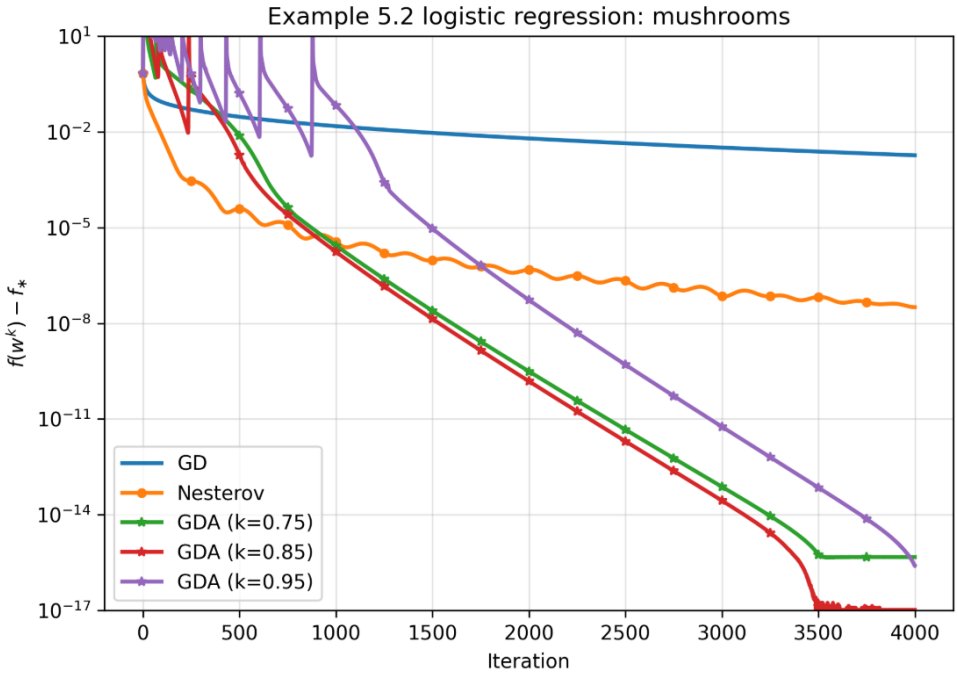
Các thực nghiệm được thực hiện với tập dữ liệu bao gồm N quan sát $(\mathbf{a}_i, b_i) \in \mathbb{R}^d \times \mathbb{R}, i = 1, \ldots, n$. Hàm mất mát cross-entropy cho hồi quy logistic đa biến được cho bởi $J(x) = -\sum_{i=1}^{N}(b_i \log(\sigma(-x^T\mathbf{a}_i)) + (1-b_i)\log(1-\sigma(-x^T\mathbf{a}_i)))$, trong đó σ là hàm sigmoid. Kết hợp với điều chuẩn ℓ_2, chúng ta có hàm mất mát được điều chuẩn $\bar{J}(x) = J(x) + \frac{1}{2N}\|x\|^2$. Hệ số Lipschitz L được ước tính bởi $\frac{1}{2N}(\|A\|^2/2+1)$, trong đó $A = (a_1^T, \ldots, a_n^T)^T$. Trong bài báo [17], các tác giả so sánh các thuật toán huấn luyện bài toán hồi quy logistic bằng cách sử dụng các tập dữ liệu Mushrooms và W8a (xem Malitsky và Mishchenko 2020) [12]. Phương pháp GDA được so sánh với thuật toán GD có kích thước bước là $1/L$ và phương pháp tăng tốc Nesterov. Chúng tôi đã tiến hành triển khai lại thí nghiệm, kết quả chi tiết được minh họa trong Hình 3 và 4. Kết quả thu được hoàn toàn tương thích với kết quả đã trình bày trong bài báo [17]. Các hình này cho thấy rằng Thuật toán GDA vượt trội hơn Thuật toán GD và phương pháp tăng tốc Nesterov về giá trị hàm mục tiêu trong các lần lặp. Đặc biệt, Hình 4 thể hiện sự thay đổi của giá trị hàm mục tiêu theo các hệ số κ khác nhau. Trong hình này, ký hiệu "GDA_0.75"tương ứng với trường hợp $\kappa = 0.75$. Hình 5 và hình inh họa sự thay đổi learning rate với tập dữ liệu Mushrooms và W8a tương ứng.

5.3 Mạng nơ-ron cho phân loại

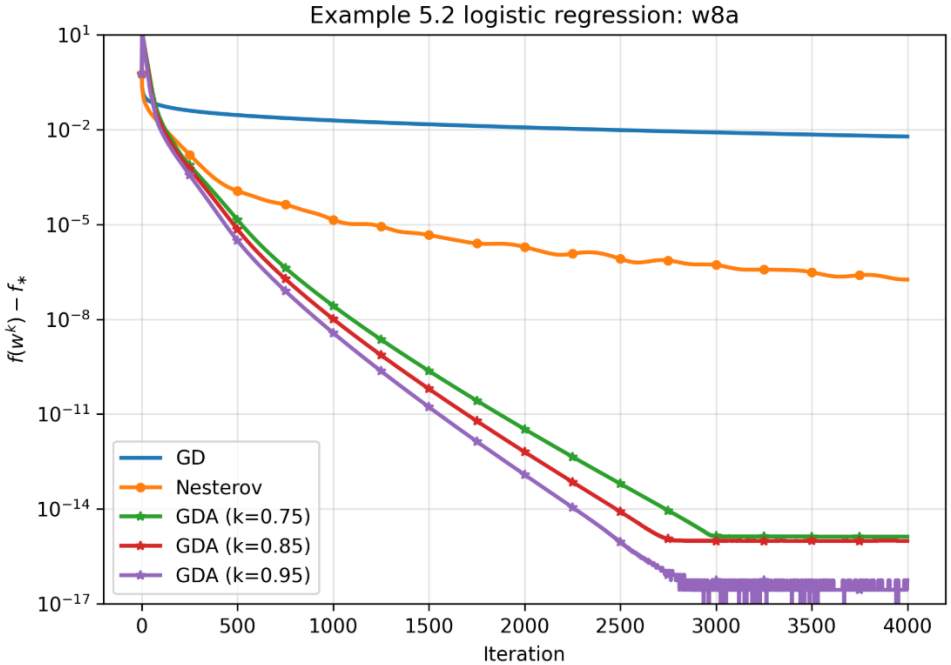
Để cung cấp một ví dụ về cách thuật toán đề xuất có thể được triển khai vào một mô hình huấn luyện mạng nơ-ron, các tác giả đã sử dụng các kiến trúc ResNet-18 tiêu chuẩn đã được triển khai trong PyTorch và huấn luyện chúng để phân loại các hình ảnh được lấy từ tập dữ liệu Cifar10 (được tải xuống tại `https://www.cs.toronto.edu/~kriz/cifar.html`), trong khi xem xét hàm mất mát cross-entropy. Trong các nghiên cứu với ResNet-18, các tác giả đã sử dụng các thiết lập mặc định của Adam cho các tham số của nó.

Để huấn luyện mô hình mạng nơ-ron này, bài báo [17] sử dụng biến thể ngẫu nhiên của phương pháp GDA (Thuật toán SGDA) để so sánh với các thuật toán Stochastic Gradient Descent (SGD). Độ chính xác kiểm thử và mức độ mất mát dữ liệu trong quá trình huấn luyện được hiển thị trong Hình 6 và 7.

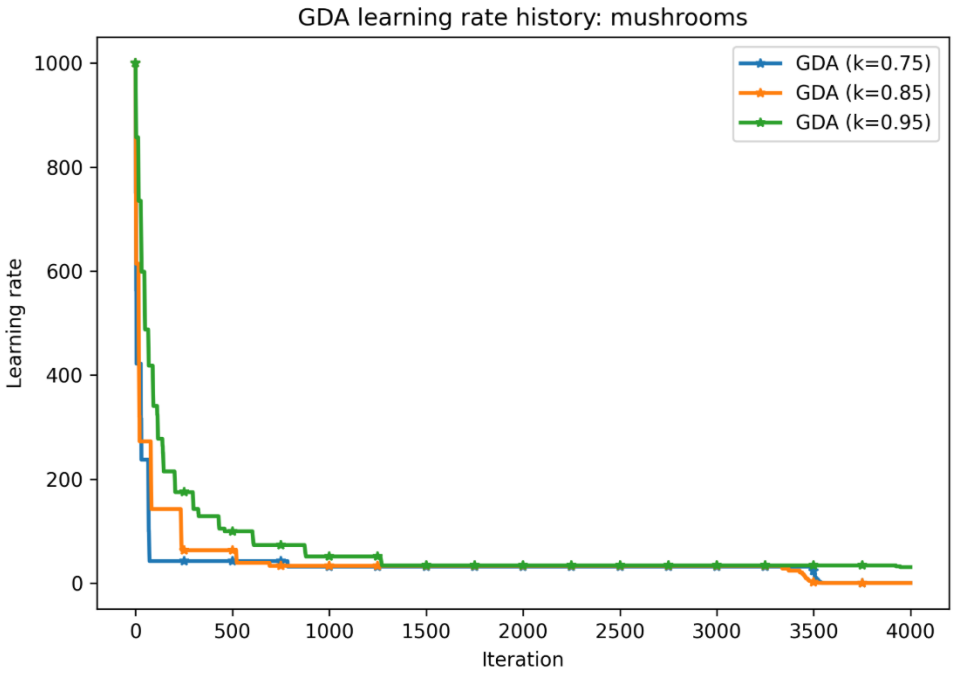
Dựa vào các biểu đồ, ta thấy thuật toán SGDA tốt hơn SGD với learning rate cố định và hội tụ nhanh hơn hẳn hai thuật toán còn lại. Tuy nhiên, thời gian chạy



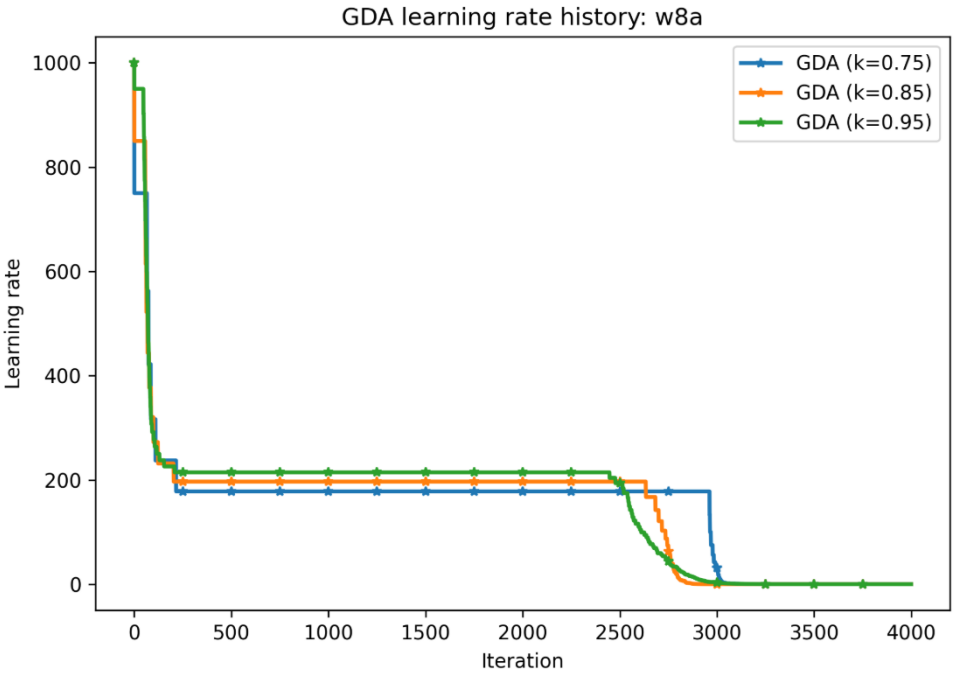
Hình 3 Kết quả tính toán cho hồi quy logistic với tập dữ liệu Mushrooms



Hình 4 Kết quả tính toán cho hồi quy logistic với tập dữ liệu W8a



Hình 5 Sự thay đổi của learning rate với tập dữ liệu Mushroom

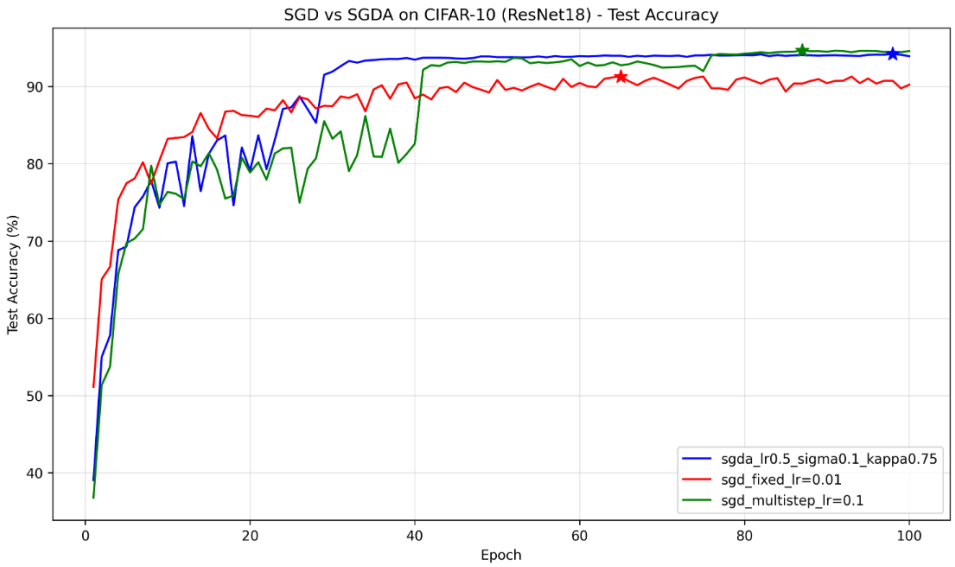


Hình 6 Sự thay đổi của learning rate với tập dữ liệu W8a

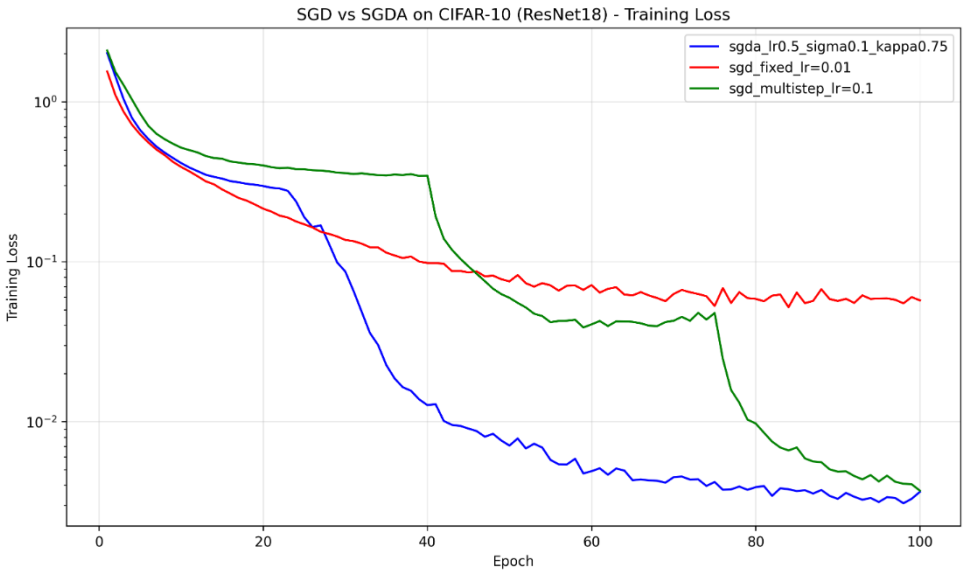
mỗi epoch tốn gấp đôi do phải tính toán trước giá trị loss, dù điều này có thể tối ưu bằng cách sử dụng lại loss trước đó.

Trong thực nghiệm này, SGDA cần 2 epoch warmup để ổn định. Nếu không có warmup, SGDA sẽ giảm learning rate xuống khoảng 0.015 ngay từ epoch đầu tiên và không giảm nữa, khiến nó hoạt động như SGD với learning rate cố định fixed_lr = 0.01. Theo chúng tôi, lý do là ở các epoch đầu, trọng số khởi tạo ngẫu nhiên làm loss biến động không ổn định, khiến điều kiện Armijo bị vi phạm liên tục và learning rate giảm nhanh. Để thực sự huấn luyện mô hình này hiệu quả, ta cần có 2 epoch để model ổn định (dùng SGD với learning rate cố định) và các epoch sau đó có thể áp dụng SGDA để thuật toán tự lựa chọn learning rate hiệu quả.

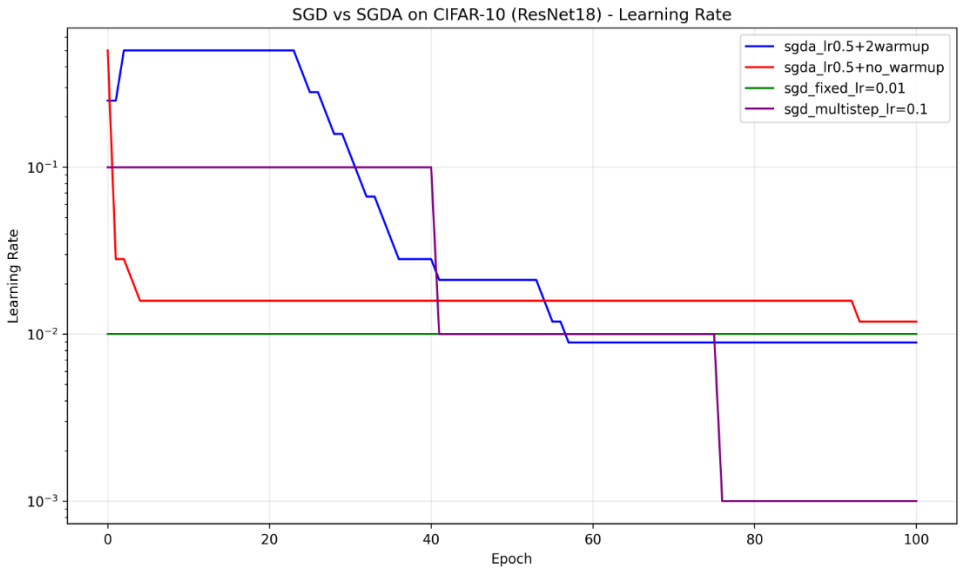
Dù kết quả cuối cùng của SGDA (test accuracy 94.21%) thấp hơn một chút so với SGD (test accuracy 94.67%) với multistep learning rate (giảm learning rate mỗi 30 epoch) , việc hội tụ sớm và khả năng tự thích nghi chứng minh SGDA hoạt động tốt trong huấn luyện các mô hình học sâu. Hình 9 và 10 lần lượt mô tả sự thay đổi của learning rate trong quá trình huấn luyện, và so sánh thời gian tính toán trung bình mỗi epoch.



Hình 7 Độ chính xác kiểm thử qua các lần lặp của mô hình ResNet-18



Hình 8 Mức độ mất mát dữ liệu trong quá trình huấn luyện qua các lần lặp của mô hình ResNet-18.



Hình 9 Sự thay đổi của learning rate trong quá trình huấn luyện mô hình ResNet18.

Hình 10 So sánh thời gian tính toán trung bình mỗi epoch (Epoch Time) trên mô hình ResNet18.

6 Kết luận

Bài báo cáo đã trình bày lại và phân tích một thuật toán gradient descent với cơ chế điều chỉnh cỡ bước tự thích nghi, cho phép áp dụng hiệu quả trên các bài toán tối ưu phi lồi và tập ràng buộc lồi, đóng nhưng không bị chặn. Thuật toán không cần tìm kiếm theo đường thẳng hay đòi hỏi biết trước hằng số Lipschitz, mà tự động điều chỉnh cỡ bước trong quá trình lặp, giúp giảm chi phí tính toán và tăng tốc độ hội tụ. Các kết quả lý thuyết và thí nghiệm số cho thấy phương pháp được đề xuất ổn định và đạt hiệu suất cao trên các bài toán lớn, bao gồm cả ứng dụng trong học máy như chọn đặc trưng có giám sát, hồi quy logistic đa biến và huấn luyện mạng nơ-ron. Nhìn chung, thuật toán cung cấp một công cụ linh hoạt và hiệu quả cho nhiều bài toán tối ưu phức tạp, đồng thời mở ra hướng nghiên cứu tiếp theo cho các bài toán tối ưu đa mục tiêu và hàm mục tiêu không trơn [16].

Tài liệu tham khảo

[1] Bauschke HH, Combettes PL. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 2011.

[2] Bian W, Ma L, Qin S, Xue X. Neural network for nonsmooth pseudoconvex optimization with general convex constraints. *Neural Networks*, 101:1–14, 2018.

[3] Boyd SP, Vandenberghe L. *Convex Optimization*. Cambridge University Press, Cambridge, 2009.

[4] Cevher V, Becker S, Schmidt M. Convex optimization for big data. *IEEE Signal Processing Magazine*, 31:32–44, 2014.

[5] Dennis JE, Schnabel RB. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, New Jersey, 1983.

[6] Ferreira OP, Sosa WS. On the frank–wolfe algorithm for non-compact constrained optimization problems. *Optimization*, 71(1):197–211, 2022.

[7] Hu Y, Li J, Yu CK. Convergence rates of subgradient methods for quasiconvex optimization problems. *Computational Optimization and Applications*, 77:183–212, 2020.

[8] Kiwiel KC. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical Programming, Series A*, 90(1):1–25, 2001.

[9] Konnov IV. Simplified versions of the conditional gradient method. *Optimization*, 67(12):2275–2290, 2018.

[10] Lan GH. *First-order and Stochastic Optimization Methods for Machine Learning*. Springer Series in the Data Sciences. Springer Nature, 2020.

[11] Liu N, Wang J, Qin S. A one-layer recurrent neural network for nonsmooth pseudoconvex optimization with quasiconvex inequality and affine equality constraints. *Neural Networks*, 147:1–14, 2022.

[12] Malitsky Y, Mishchenko K. Adaptive gradient descent without descent. In *Proceedings of Machine Learning Research*, volume 119, pages 6702–6712, 2020.

[13] Mangasarian O. Pseudo-convex functions. *SIAM Journal on Control*, 8:281–289, 1965.

[14] Nesterov Y. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87. Springer Science & Business Media, 2013.

[15] Rockafellar RT. *Convex Analysis*. Princeton University Press, Princeton, 1970.

[16] Thang TN, Solanki VK, Dao TA, Anh NTN, Hai PV. A monotonic optimization approach for solving strictly quasiconvex multiobjective programming problems. *Journal of Intelligent & Fuzzy Systems*, 38:6053–6063, 2020.

[17] Trần Ngọc Thăng, Trịnh Ngọc Hải. Self-adaptive algorithms for quasiconvex programming and applications to machine learning. *Computational and Applied Mathematics*, 43(249), 2024.

[18] Wang Y, Li X, Wang J. A neurodynamic optimization approach to supervised feature selection via fractional programming. *Neural Networks*, 136:194–206, 2021.

[19] Xu HK. Iterative algorithms for nonlinear operators. *Journal of the London Mathematical Society*, 66:240–256, 2002.

[20] Yu CK, Hu Y, Yang X, Choy SK. Abstract convergence theorem for quasiconvex optimization problems with applications. *Optimization*, 68(7):1289–1304, 2019.