

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

TIỂU LUẬN

Tính toán song song thuật toán Gradient Descent

HOÀNG PHI LONG

philonghust@gmail.com

Ngành: Toán Tin

Giảng viên hướng dẫn: TS. Đoàn Duy Trung Chữ kí của GVHD

Viện: Toán ứng dụng và Tin học

HÀ NỘI, 7/2021

Lời cảm ơn

Tác giả gửi lời cảm ơn sâu sắc tới **TS. Đoàn Duy Trung**, người thầy đã tận tình chỉ bảo với cách dạy sáng tạo, hiện đại và tạo môi trường học năng động cho lớp **Tính Toán Song Song**. Không có hướng dẫn của thầy, tiểu luận sẽ không thể hoàn thiện.

Tác giả xin gửi lời cảm ơn đến Viện Toán ứng dụng và Tin học, Trường Đại học Bách Khoa Hà Nội đã cung cấp những kiến thức giúp cho tác giả có những điều kiện thuận lợi để hoàn thành tiểu luận này.

Tác giả đặc biệt bạn Nguyễn Hải Đăng, Đỗ Quang Hùng, và Nguyễn Kim Long kì vừa rồi đã đồng hành cùng tác giả để giúp đỡ nhau học tập và giải quyết các bài tập lớn.

Tác giả xin chân thành cảm ơn!

Tóm tắt nội dung tiểu luận

Trong Machine Learning nói riêng và Toán Tối Ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó (hàm mất mát). Nhìn chung, việc tìm global minimum của các hàm mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm local minimum, và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.

Các điểm local minimum là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm local minimum đó vào hàm số rồi tìm điểm làm cho hàm có giá trị nhỏ nhất. Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của dạng của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc từ việc có quá nhiều điểm dữ liệu. Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là gần với nghiệm của bài toán, sau đó dùng một phép toán lặp để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (viết gọn là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

Tuy nhiên, với sự phát triển của Internet ngày nay, kích thước dữ liệu chúng ta cần xử lý là rất lớn, khiến cho thời gian tính toán rất lâu. Trong nội dung của bài tiểu luận này, tác giả sẽ áp dụng những kĩ thuật tính toán song song đã học để cải thiện tốc độ tính toán của thuật toán GD (cụ thể ở trong bài toán xây dựng mô hình hồi quy tuyến tính Linear Regression).

Từ khóa: *Gradient Descent, Linear Regression, Computing Parallel.*

Hà Nội, ngày 20 tháng 07 năm 2021

Học viên thực hiện

Hoàng Phi Long

MỤC LỤC

1	CƠ SỞ LÝ THUYẾT	1
1.1	Bài toán quy hoạch phi tuyến không ràng buộc	1
1.1.1	Điều kiện tối ưu	1
1.1.2	Hướng giảm	1
1.2	Gradient Descent	2
1.2.1	Thuật toán	2
1.2.2	Lựa chọn learning rate	3
1.3	Mô hình hồi quy tuyến tính	4
1.3.1	Giới thiệu về mô hình hồi quy tuyến tính	4
1.3.2	Ước lượng bình phương cực tiểu	6
1.3.3	Áp dụng Gradient Descent để tìm nghiệm xấp xỉ cho mô hình hồi quy tuyến tính	6
2	TÍNH TOÁN SONG SONG	
	THUẬT TOÁN GRADIENT DESCENT	9
2.1	Bài toán nhân hai ma trận	9
2.2	Nhân song song hai ma trận	10
2.2.1	Phân chia dữ liệu theo dải băng	10
2.2.2	Tính toán song song nhân ma trận	11
3	CÀI ĐẶT CHƯƠNG TRÌNH	12
3.1	Dữ liệu	12
3.1.1	Tạo dữ liệu	12
3.1.2	Cấu trúc của file dữ liệu	12
3.2	Cài đặt	13
3.2.1	Hàm tính chuẩn vector	13
3.2.2	Hàm nhân hai ma trận	14
3.2.3	Hàm sử dụng Gradient Descent để huấn luyện mô hình hồi quy tuyến tính	15

3.2.4	Toàn bộ chương trình	16
4	ĐÁNH GIÁ KẾT QUẢ	22
4.1	Kiểm tra tính đúng đắn của chương trình	22
4.2	So sánh chương trình tuần tự và song song	23
5	KẾT LUẬN	24
	TÀI LIỆU THAM KHẢO	24

DANH MỤC HÌNH VẼ

Hình 1.1.	Các tham số θ được tinh chỉnh để giảm thiểu hàm mất mát	2
Hình 1.2.	Learning rate quá nhỏ	3
Hình 1.3.	Learning rate quá lớn	3
Hình 1.4.	Cạm bẫy của GD.	4
Hình 1.5.	Mô hình hồi quy tuyến tính	5
Hình 1.6.	Ước lượng bình cực tiểu là nghiệm tối ưu cho hàm MSE . .	6
Hình 1.7.	Gradient Descent với các learning khác nhau	8
Hình 2.1.	Minh họa việc tính toán song song ma trận	11
Hình 4.1.	Kết quả chương trình khá chính xác với $n = 5000, p = 10$. .	22

DANH MỤC BẢNG BIỂU

Bảng 4.1	So sánh chương trình tuần tự và song song	23
----------	---	----

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

Từ viết tắt	Ý nghĩa
-------------	---------

GD	Gradient Descent
----	------------------

MSE	Mean Square Error
-----	-------------------

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1 Bài toán quy hoạch phi tuyến không ràng buộc

Bài toán quy hoạch phi tuyến không ràng buộc được phát biểu như sau: [1]

$$\min f(\theta), \text{ với } \theta \in \mathbb{R}^n \quad (P^{krb})$$

trong đó $f: \mathbb{R}^n \rightarrow \mathbb{R}$ là hàm phi tuyến.

1.1.1 Điều kiện tối ưu

Mệnh đề 1.1. Cho hàm f xác định, khả vi trên \mathbb{R}^n . Nếu $x \in \mathbb{R}^n$ là nghiệm cực tiểu địa phương của bài toán (P^{krb}) thì $\nabla f(\theta) = 0$

Mệnh đề 1.2. Giả sử f là hàm lồi khả vi trên \mathbb{R}^n . Khi đó $\theta^* \in \mathbb{R}^n$ là nghiệm toàn cục của bài toán khi và chỉ khi $\nabla f(\theta^*) = 0$

1.1.2 Hướng giảm

Định nghĩa 1.1. Cho $\theta^0 \in \mathbb{R}^n$. Ta gọi $d \in \mathbb{R}^n$ là hướng giảm của hàm f tại θ^0 nếu tồn tại $\epsilon > 0$ sao cho với mọi t thỏa mãn $0 < t < \epsilon$ ta có $f(\theta^0 + td) < f(\theta^0)$.

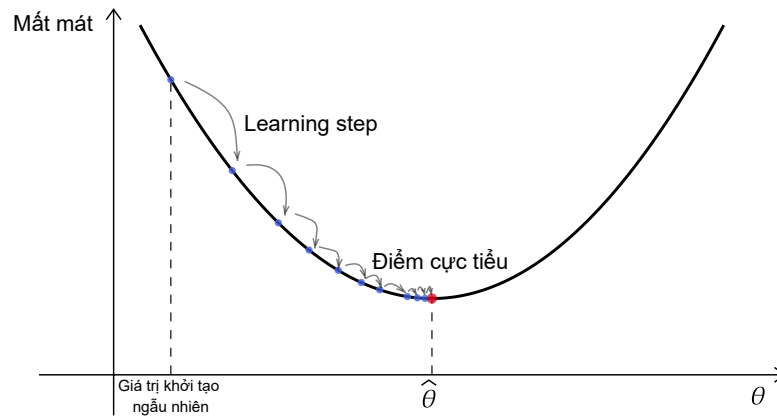
Mệnh đề 1.3. Giả sử hàm f là hàm khả vi trên \mathbb{R}^n và $\nabla f(\theta^0) \neq 0$. Trong các hướng giảm d của hàm f tại θ^0 có $\|d\|$ thì hàm f giảm nhanh nhất theo hướng $d = -\frac{\nabla f(\theta^0)}{\|\nabla f(\theta^0)\|}$.

1.2 Gradient Descent

1.2.1 Thuật toán

Gradient Descent (GD) là một thuật toán tối ưu hóa chung có khả năng tìm ra giải pháp tối ưu cho rất nhiều bài toán khác nhau (bao gồm bài toán P^{krb}). Ý tưởng chung của GD là tinh chỉnh các tham số θ lặp đi lặp lại để tối ưu hàm chi phí.

Giả sử bạn bị lạc ở trên một ngọn núi phủ sương, và bạn chỉ có thể cảm nhận được độ dốc của mặt đất bên dưới chân bạn. Một chiến thuật để đi xuống chân núi nhanh chóng chính là đi theo hướng dốc nhất. Đó chính xác là cái mà GD làm: thuật toán tính hướng giảm $\nabla f(\theta^0)$, sau đó đi ngược hướng gradient. Khi $\nabla f(\theta^0) = 0$, ta thu được một điểm cực tiểu địa phương.



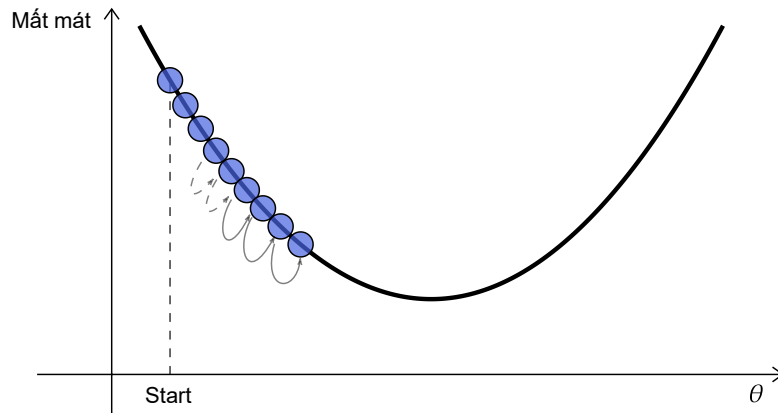
Hình 1.1. Các tham số θ được tinh chỉnh để giảm thiểu hàm mất mát

Thuật toán 1.1. Thuật toán GD có thể được trình bày như sau:

1. Khởi tạo: Chọn trước số $\epsilon > 0$ đủ nhỏ, độ dài bước η (learning rate) hợp lý (sẽ trình bày kĩ hơn ở mục 1.2.2). Xuất phát từ một điểm tùy ý $\theta \in \mathbb{R}^n$ có $\nabla f(\theta) \neq 0$.
2. Tính $\nabla f(\theta)$. Nếu $\|\nabla f(\theta)\| \leq \epsilon$ thì chuyển sang bước 5.
3. Tính $\theta^{(next\ step)} := \theta - \eta \nabla f(\theta)$
4. Quay lại bước 2.
5. Đưa ra nghiệm cực tiểu θ và kết thúc thuật toán.

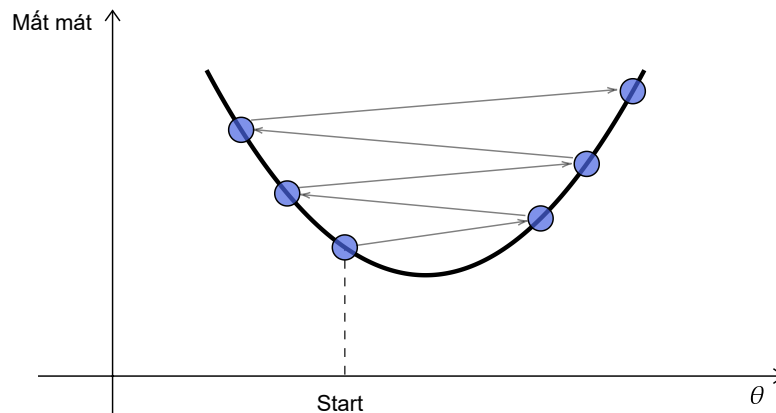
1.2.2 Lựa chọn learning rate

Một tham số rất quan trọng trong GD là độ dài bước, hay còn gọi là *learning rate*. Nếu learning rate quá nhỏ, thuật toán 1.1 sẽ phải lặp rất nhiều lần để hội tụ, khiến mất rất nhiều thời gian.



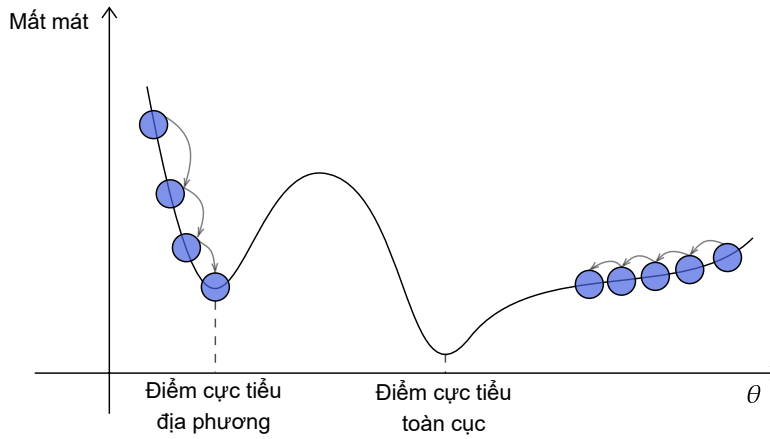
Hình 1.2. Learning rate quá nhỏ

Mặt khác, nếu learning rate quá lớn, ta có thể đi qua điểm cực tiểu, điều này có thể khiến cho thuật toán phân kì và gây ra hiện tượng tràn số.



Hình 1.3. Learning rate quá lớn

Lưu ý, không phải lúc nào hàm mất mát đều có hình dạng đơn giản giống các hình trên. Chúng có thể có hình dạng bất kỳ và độ phức tạp khác nhau, khiến cho việc hội tụ đến điểm cực tiểu toàn cục gặp nhiều khó khăn hơn.



Hình 1.4. Cạm bẫy của GD

Hình 1.4. cho ta thấy 2 thử thách chính của GD: [2]

- Nếu điểm khởi đầu ngẫu nhiên xuất phát từ phía bên trái, thuật toán sẽ hội tụ tại điểm cực tiểu địa phương.
- Nếu điểm khởi đầu ngẫu nhiên xuất phát từ phía bên phải, thuật toán sẽ mất rất nhiều thời gian để hội tụ đến điểm cực tiểu địa phương. Và nếu thuật toán dừng quá sớm, chúng ta sẽ không bao giờ chạm tới điểm cực tiểu toàn cục.

1.3 Mô hình hồi quy tuyến tính

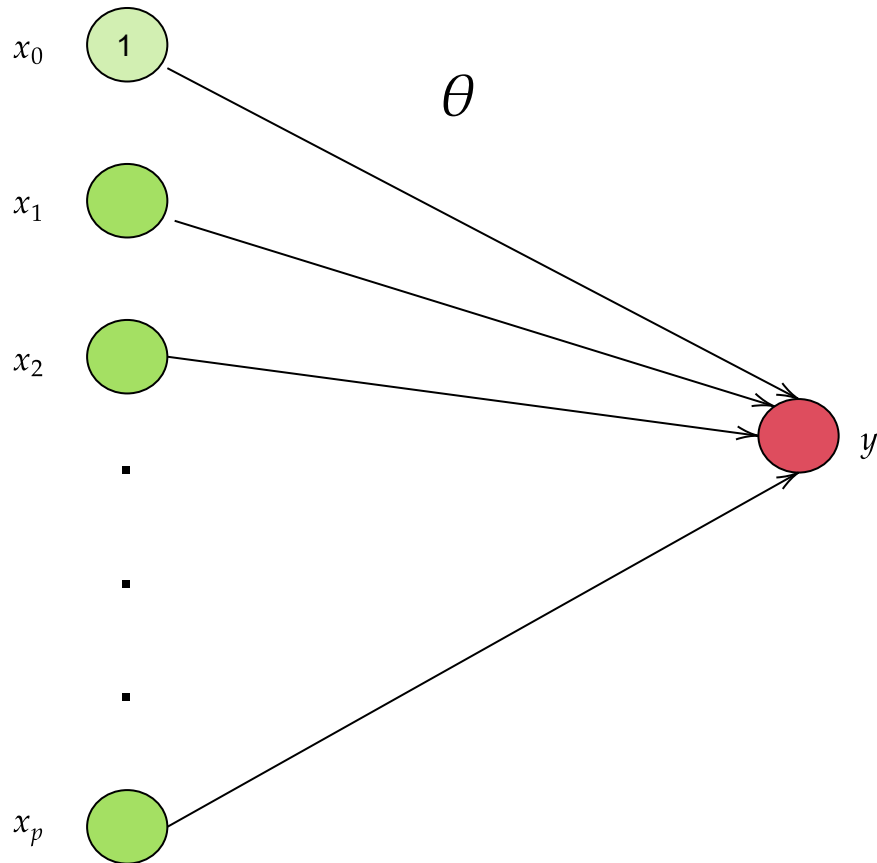
1.3.1 Giới thiệu về mô hình hồi quy tuyến tính

Mô hình toán học

Giả sử ta có tập mẫu gồm n quan sát p biến độc lập X_1, X_2, \dots, X_k dùng để dự báo và y là biến phụ thuộc cần dự báo. Sự phụ thuộc của y theo X_1, X_2, \dots, X_k thường phức tạp. Tuy nhiên có một số trường hợp sự phụ thuộc đó khá đơn giản. Mô hình hồi quy tuyến tính khẳng định y phụ thuộc tuyến tính vào các X_i , nghĩa là:

$$y = \theta_0 + \theta_1 X_1 + \dots + \theta_p X_p + \epsilon \quad (1.3.1)$$

trong đó $\theta_i, i = \overline{1, p}$ là các hệ số chưa biết, còn ϵ là các sai số ngẫu nhiên tuân theo phân phối chuẩn. [3]



Hình 1.5. Mô hình hồi quy tuyến tính

Mô hình hồi quy tuyến tính có thể viết gọn:

$$y = \theta X + \epsilon \quad (1.3.2)$$

trong đó $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times (p+1)}$, $\theta \in \mathbb{R}^{p+1}$, $\epsilon \in \mathbb{R}^n$

Hàm mất mát

Hàm mất mát của mô hình hồi quy tuyến tính là hàm *Mean Square Error*:

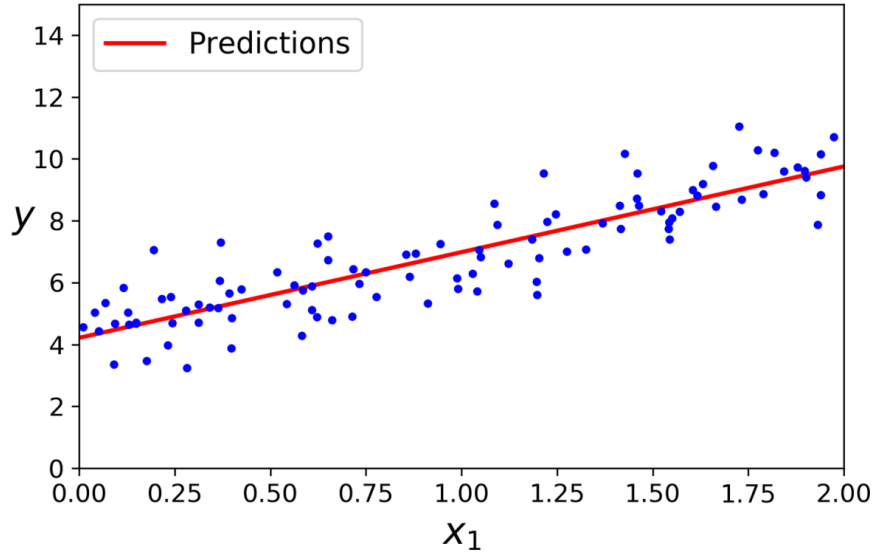
$$\text{MSE}(\theta) = \frac{1}{n} \sum_{i=0}^{n-1} (\theta^T x^{(i)} - y^{(i)})^2 \quad (1.3.3)$$

Hàm mất mát là một đại lượng để đo lường độ chính xác của mô hình. Do đó, để huấn luyện mô hình hồi quy tuyến tính, ta cần tìm giá trị θ sao cho tối ưu hàm MSE.

1.3.2 Ước lượng bình phương cực tiểu

Mệnh đề 1.4. Nếu ma trận X có hạng $p + 1 \leq n$ thì ước lượng bình phương cực tiểu là nghiệm toàn cục của hàm mất mát MSE :

$$\hat{\theta} = (X^T X)^{-1} X^T y \quad (1.3.4)$$



Hình 1.6. Ước lượng bình phương cực tiểu là giá trị tối ưu cho hàm MS.

Độ phức tạp tính toán

Ước lượng bình phương cực tiểu yêu cầu tính toán $(X^T X)^{-1}$, là ma trận có kích thước $(p + 1) \times (p + 1)$. Độ phức tạp tính toán nghịch đảo của ma trận thông thường nằm trong khoảng $O(p^{2.4})$ đến $O(p^3)$ [2]. Vì vậy, ước lượng bình phương cực tiểu thường tính toán rất lâu nếu kích thước dữ liệu lớn ($p > 10000$).

1.3.3 Áp dụng Gradient Descent để tìm nghiệm xấp xỉ cho mô hình hồi quy tuyến tính

Như đã đề cập ở phần 1.3.2, độ phức tạp tính của ước lượng bình phương cực tiểu là vô cùng lớn khi kích thước dữ liệu tăng, và với các mô hình phức tạp như Deep Neuron Network, việc tìm ra chính xác nghiệm tối ưu gần như là không thể. Vì vậy, ta có thể tìm nghiệm xấp xỉ để thay thế nghiệm chính xác bằng phương pháp Gradient Descent để giảm thiểu độ phức tạp tính toán.

Một số công thức

Về cơ bản, thuật toán áp dụng để tìm nghiệm xấp xỉ cho mô hình hồi quy tuyến tính giống với thuật toán 1.1, với hàm mục tiêu là hàm mất mát MSE (1.3.3). May mắn là hàm MSE là hàm lồi, vì vậy vậy, nghiệm cực tiểu địa phương của bài toán cũng là nghiệm cực tiểu toàn cục (theo mệnh đề 1.2).

Ta có đạo hàm riêng của hàm MSE là:

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{n} \sum_{i=0}^{n-1} (\theta^T x^{(i)} - y^{(i)}) x_j^{(i)} \quad (1.3.5)$$

với n là số lượng quan sát và $x^{(i)}, y^{(i)}$ là quan sát và biến dự đoán thứ $i + 1$ trong tập mẫu, $x_j^{(i)}$ là biến thứ j của quan sát thứ $i + 1$.

Vector gradient $\nabla \text{MSE}(\theta)$, gồm đạo hàm riêng theo tất cả các biến:

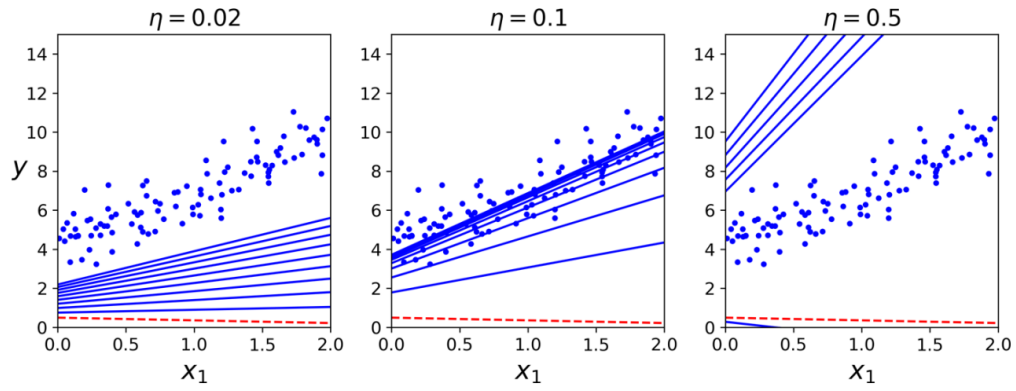
$$\nabla \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{n} X^T (X\theta - y) \quad (1.3.6)$$

Khi đã có vector $\nabla \text{MSE}(\theta)$, ta sẽ đi ngược hướng gradient để tiến đến nghiệm tối ưu:

$$\theta^{(\text{next step})} := \theta - \eta \nabla f(\theta) \quad (1.3.7)$$

với η là learning rate.

Nếu learning rate η được lựa chọn hợp lý, ta sẽ thu được kết quả gần chính xác với nghiệm ước lượng bình phương cực tiểu. Tuy nhiên, nếu chúng ta lựa chọn learning rate không hợp lý sẽ dẫn đến những hậu quả khôn lường. Hình 1.7. biểu diễn 10 bước đầu của Gradient Descent với 3 learning rate khác nhau (đường gạch ngang màu đỏ là điểm bắt đầu).



Hình 1.7. Gradient Descent với các learning khác nhau

Hình bên trái, learning rate quá nhỏ, thuật toán sẽ hội tụ đến nghiệm tối ưu, nhưng sẽ mất rất nhiều thời gian. Hình ở giữa, learning có vẻ hợp lý, chỉ sau một vài lần lặp, thuật toán đã hội tụ đến nghiệm tối ưu. Hình bên phải, learning rate quá cáo, thuật toán phân kì, "nhảy"qua nghiệm tối ưu và ngày càng xa nghiệm tối ưu qua mỗi lần lặp.

Độ phức tạp tính toán

Với hàm mất mát là hàm lồi và độ dốc của nó không thay đổi đột ngột, thuật toán Gradient Descent với learning rate hợp lý sẽ hội tụ tới nghiệm tối ưu, với độ phức tạp tính toán là $O(1/\epsilon)$ (ϵ là độ lớn tối thiểu để coi như $\nabla f(\theta) = 0$).

CHƯƠNG 2. TÍNH TOÁN SONG SONG

THUẬT TOÁN GRADIENT DESCENT

Ta có thể thấy, khi áp dụng thuật toán Gradient Descent, có một số công việc ta có thể thực hiện song song như nhân ma trận (công thức 1.3.6 và 1.3.7) và tính chuẩn $\|\nabla \text{MSE}(\theta)\|$. Việc thực hiện song song tính chuẩn $\|\nabla \text{MSE}(\theta)\|$ khá đơn giản. Vì vậy, tác giả sẽ trình bày chi tiết việc tính toán song song nhân ma trận.

2.1 Bài toán nhân hai ma trận

Bài toán nhân nhân ma trận A cỡ $n \times p$ và ma trận B cỡ $p \times m$, kết quả nhận được là 1 ma trận cỡ $n \times m$, tức là

$$A_{n \times p} \cdot B_{p \times m} = C_{n \times m} \quad (2.1.1)$$

$$c_{ij} = \sum_{k=0}^{p-1} a_{ik} b_{kj}, \text{ với } i = \overline{0, n-1}, j = \overline{0, m-1} \quad (2.1.2)$$

Trong đó:

- c_{ij} là phần tử hàng i , cột j là của ma trận kết quả C .
- a_{ik} là phần tử hàng i , cột k là của ma trận kết quả A .
- b_{ki} là phần tử hàng k , cột j là của ma trận kết quả B .

Ví dụ nhân ma trận $A_{3 \times 4}$ và ma trận $B_{4 \times 2}$ ta được ma trận $C_{3 \times 2}$ như sau:

$$\begin{pmatrix} 3 & 2 & 0 & -1 \\ 5 & -2 & 1 & 1 \\ 1 & 0 & -1 & -1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 8 & 5 \\ -6 & -1 \end{pmatrix} \quad (2.1.3)$$

2.2 Nhân song song hai ma trận

Bài toán nhân 2 ma trận được đặc trưng bởi sự lặp lại trong quá trình tính toán với tất cả các phần tử trong ma trận. Vì vậy, lựa chọn song song dữ liệu khi thực hiện tính toán sẽ là lựa chọn khôn ngoan hơn việc phân chia song song các hoạt động của ma trận được phân bố vào các processor trong hệ thống tính toán. Có rất nhiều phương pháp để chia ma trận để tính toán, tuy nhiên có 2 phương pháp phổ biến và sử dụng rộng rãi là phương pháp phân chia dữ liệu theo dải băng (theo chiều ngang hoặc chiều dọc) và phân đoạn theo hình khối [4]. Trong tiểu luận này, tác giả sẽ sử dụng phương pháp phân chia dữ liệu theo dải băng.

2.2.1 Phân chia dữ liệu theo dải băng

Khi thực hiện phương pháp này, mỗi processor sẽ được phân chia theo nhóm các hàng (theo hàng ngang) hoặc nhóm các cột (theo cột dọc) của ma trận.

Để thực hiện phân chia theo chiều ngang bởi các hàng của ma trận, ta sẽ chia ma trận A như sau:

$$A = (A_0, A_1, \dots, A_{n-1})^T \quad (2.2.1)$$

trong đó:

- $A_i = (a_{i_0}, a_{i_1}, \dots, a_{i_{n-1}})^T$ được gọi là hàng thức thứ i của ma trận A (kích thước $n \times p$).
- $a_{ij} = A_{ij}$ là phần tử hàng i cột j của ma trận A .
- $i_j = ip + j, 0 \leq j < p, 0 \leq i < n$.

Để thực hiện phân chia theo chiều dọc bởi các cột của ma trận, ta sẽ chia ma trận B như sau:

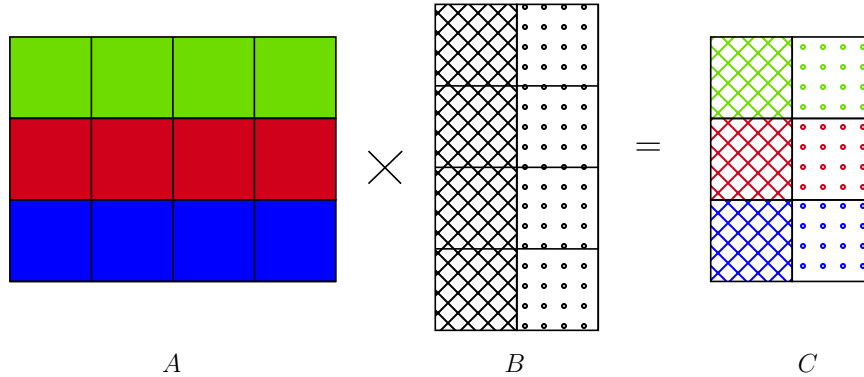
$$B = (B_0, B_1, \dots, B_{n-1}) \quad (2.2.2)$$

trong đó:

- $B_j = (b_{0_j}, b_{1_j}, \dots, b_{(p-1)_j})$ được gọi là cột thức thứ j của ma trận B (kích thước $p \times m$).
- $b_{i_j} = b_{ij}$ là phần tử hàng i cột j của ma trận B .
- $i_j = jm + i, 0 \leq j < p, 0 \leq i < m$.

2.2.2 Tính toán song song nhân ma trận

Khi tính toán song ma trận, ta sẽ nhân song song các hàng ở ma trận A và các cột ở ma trận B , được minh họa giống sơ đồ bên dưới.



Hình 2.1. Minh họa việc tính toán song song ma trận

CHƯƠNG 3. CÀI ĐẶT CHƯƠNG TRÌNH

3.1 Dữ liệu

3.1.1 Tạo dữ liệu

Mục đích của bài tiểu luận là sử dụng thuật toán GD để huấn luyện mô hình hồi quy tuyến tính và áp dụng tính toán song song để giảm thiểu thời gian huấn luyện, vì vậy để trực quan và thực nghiệm tính đúng đắn của nghiên cứu, tác giả sẽ tạo một tập dữ liệu chứa các giá trị ngẫu nhiên gồm X, y sao cho:

$$y_i = \sum_{j=0}^p j \cdot x_{ij} + \epsilon_i \quad (3.1.1)$$

với x_{ij} là giá trị phần tử hàng i cột j của ma trận X , và y_i là biến quan sát thứ i , ϵ_i là thiên lệch ngẫu nhiên tuân theo phân phối chuẩn của quan sát thứ i .

Như vậy, ta đã biết rằng giá trị chính xác của $\theta_j = j$.

Ngoài ra để tiện so sánh giữa chương trình tính toán song song và tuần tự, tác giả cũng khởi tạo sẵn giá trị xuất phát cho vector θ và lưu vào file **theta_p.txt** (đối với tập dữ liệu có p biến).

3.1.2 Cấu trúc của file dữ liệu

Dữ liệu hiện đã có sẵn tại link **Github** của tác giả.

Hướng dẫn cách đọc dữ liệu trong file:

- Với file tên là **n_p.txt**, dòng đầu gồm 2 số nguyên n, p là số lượng quan sát và số biến của tập mẫu. n dòng tiếp theo, mỗi dòng chứa $p + 1$ số thực, cụ thể với dòng thứ i thì p số thực đầu là các giá trị $(x_{i1}, x_{i2}, \dots, x_{ip})$, số thực còn lại là giá trị của biến dự báo y_i tương ứng.
- Với file tên là **theta_p.txt**, file gồm p dòng, dòng thứ i là giá trị khởi tạo ban đầu của hệ số θ_i .

3.2 Cài đặt

Có rất nhiều công cụ hỗ trợ lập trình song song, tuy nhiên tác giả sẽ sử dụng API OpenMP để cài đặt chương trình trong tiểu luận này, vì:

- OpenMP là chuẩn hoàn chỉnh và được công nhận trên thực tế.
- Hiệu suất và khả năng mở rộng tốt.
- Tính khả chuyển cao: Chương trình viết ra có thể dịch bằng nhiều trình dịch khác nhau.
- Dễ sử dụng, đơn giản, ít các chỉ thị.
- Cho phép song song hóa nhiều chương trình tuần tự

3.2.1 Hàm tính chuẩn vector

Đối số đầu vào là độ dài và tọa độ của vector cần tính. Chuẩn được sử dụng ở đây là chuẩn 2. Đối với hàm `norm_parallel`, ta sẽ sử dụng khối chia sẻ `for` và 1 biến `reduction(+:norm)` để phân chia công việc đến các luồng và tổng hợp kết quả chuẩn cần tính.

```
1  double norm_parallel(int p, double* &gradient)
2  {
3      int i;
4      double norm=0;
5
6      # pragma omp parallel for reduction(+:norm)
7      for (i=0; i<p+1; i++)
8      {
9          norm = gradient[i]*gradient[i];
10     }
11     return sqrt(norm);
12 }
13
14 double norm_serial(int p, double* &gradient)
15 {
16     int i;
17     double norm=0;
18
19     for (i=0; i<p+1; i++)
20     {
```

```

21         norm += gradient[i]*gradient[i];
22     }
23     return sqrt(norm);
24 }

```

Mã 3.1: Hàm tính chuẩn vector

3.2.2 Hàm nhân hai ma trận

Thực hiện song song nhân ma trận và véc tơ với kiểu phân chia dữ liệu theo mô hình dải băng theo từng hàng. Hàm này cho phép nhân từng nhóm hàng của ma trận A với từng nhóm cột của ma trận B sử dụng một vài luồng song song. Mỗi luồng sẽ thực hiện tính toán và lưu kết quả vào ma trận C .

```

1  int matmul_parallel(int n, int p, int m, double* A, double* B, double* C)
2  {
3      int i,j,k;
4      #pragma omp parallel shared(A,B,C) private(i,j,k)
5      {
6          #pragma omp for schedule(static)
7          for (i=0; i<n; i++)
8              for (j=0; j<m; j++)
9              {
10                 C[i*m+j] = 0;
11                 for (k=0; k<p; k++)
12                     C[i*m+j] += A[i*p+k]*B[k*m+j];
13             }
14     }
15     return 0;
16 }
17
18 int matmul_serial(int n, int p, int m, double* A, double* B, double* C)
19 {
20     int i,j,k;
21     {
22         for (i=0; i<n; i++)
23             for (j=0; j<m; j++)
24             {
25                 C[i*m+j] = 0;
26                 for (k=0; k<p; k++)
27                     C[i*m+j] += A[i*p+k]*B[k*m+j];

```

```

28         }
29     }
30     return 0;
31 }

```

Mã 3.2: Hàm nhân hai ma trận

3.2.3 Hàm sử dụng Gradient Descent để huấn luyện mô hình hồi quy tuyến tính

Về cơ bản, hàm được lập trình theo thuật toán 1.1, với hàm song song thì việc tính toán ma trận và tính chuẩn được thực hiện song song. Trong quá trình cài đặt, tác giả đã lựa chọn cố định $\text{learning rate} = 0.1$ và $\epsilon = 10^{-10}$.

```

1 void gradient_descent_parallel(double* &X, double* &X_T, double* &y, double*
   &theta, double* &gradient)
2 {
3     double* X_theta;
4     int i;
5     double learning_rate=0.1, eps=0.0000000001;
6     X_theta = new double[n*1];
7     do
8     {
9
10         matmul_parallel(n, p+1, 1, X, theta, X_theta);
11         # pragma omp parallel shared (X_theta, n) private (i)
12         # pragma omp for
13         for (i=0; i<n; i++)
14             X_theta[i] -= y[i];
15
16         matmul_parallel(p+1, n, 1, X_T, X_theta, gradient);
17
18         # pragma omp parallel shared (gradient, p) private (i)
19         # pragma omp for
20         for (i=0; i<p+1; i++)
21             gradient[i] = gradient[i]*2/n;
22
23         # pragma omp parallel shared (gradient, p) private (i)
24         # pragma omp for
25         for (int i = 0; i<p+1; i++)
26             theta[i] = theta[i] - learning_rate*gradient[i];

```

```

27     }
28     while (norm_parallel(p, gradient)>eps);
29     delete[] X_theta;
30 }
31
32 void gradient_descent_serial(double* &X, double* &X_T, double* &y, double* &
    theta, double* &gradient)
33 {
34     double* X_theta;
35     int i;
36     double learning_rate=0.1, eps=0.0000000001;
37     X_theta = new double[n*1];
38     do
39     {
40         matmul_serial(n, p+1, 1, X, theta, X_theta);
41         for (i=0; i<n; i++)
42             X_theta[i] -= y[i];
43
44         matmul_serial(p+1, n, 1, X_T, X_theta, gradient);
45
46         for (i=0; i<p+1; i++)
47             gradient[i] = gradient[i]*2/n;
48
49         for (int i = 0; i<p+1; i++)
50             theta[i] = theta[i] - learning_rate*gradient[i];
51     }
52     while (norm_parallel(p, gradient)>eps);
53     delete[] X_theta;
54 }

```

Mã 3.3: Gradient Descent

3.2.4 Toàn bộ chương trình

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <conio.h>
4  #include <omp.h>
5  #include <time.h>
6  #include <math.h>
7  int n, p;

```



```

8
9 void read_data(double* &X, double* &X_T, double* &y, double* &theta, double*
    &gradient)
10 {
11     FILE *f;
12     f = fopen("5000_10.txt", "r");
13     fscanf(f, "%d %d", &n, &p);
14     printf("Tinh toan voi du lieu n=%d va p%d\n", n, p);
15     X = new double[n * (p+1)];
16     X_T = new double[(p+1) * n];
17     y = new double[n * 1];
18     theta = new double[(p+1) * 1];
19     gradient = new double[(p+1) * 1];
20     for (int i=0; i<n; i++)
21     {
22         X[i*(p+1)] = 1.0;
23         for (int j=1; j<p+2; j++)
24         {
25             if (j<p+1) fscanf(f, "%lf", &X[i*(p+1)+j]);
26             else fscanf(f, "%lf\n", &y[i]);
27         }
28     }
29
30     for (int i=0; i<p+1; i++)
31         for (int j=0; j<n; j++)
32             X_T[i*n+j] = X[j*(p+1)+i];
33
34     fclose(f);
35     f = fopen("theta_10.txt", "r");
36     for (int i=0; i<p+1; i++)
37     {
38         if (i<p) fscanf(f, "%lf\n", &theta[i]);
39         else fscanf(f, "%lf", &theta[i]);
40     }
41     fclose(f);
42 }
43
44 int matmul_parallel(int n, int p, int m, double* A, double* B, double* C)
45 {
46     int i,j,k;
47     #pragma omp parallel shared(A,B,C) private(i,j,k)
48     {
49     #pragma omp for schedule(static)

```

```

50     for (i=0; i<n; i++)
51         for (j=0; j<m; j++)
52             {
53                 C[i*m+j] = 0;
54                 for (k=0; k<p; k++)
55                     C[i*m+j] += A[i*p+k]*B[k*m+j];
56             }
57     }
58     return 0;
59 }
60
61 int matmul_serial(int n, int p, int m, double* A, double* B, double* C)
62 {
63     int i,j,k;
64     {
65         for (i=0; i<n; i++)
66             for (j=0; j<m; j++)
67                 {
68                     C[i*m+j] = 0;
69                     for (k=0; k<p; k++)
70                         C[i*m+j] += A[i*p+k]*B[k*m+j];
71                 }
72     }
73     return 0;
74 }
75
76 double norm_parallel(int p, double* &gradient)
77 {
78     int i;
79     double norm=0;
80
81
82     # pragma omp parallel for reduction(+:norm)
83     for (i=0; i<p+1; i++)
84     {
85         norm = gradient[i]*gradient[i];
86     }
87
88     return sqrt(norm);
89
90 }
91
92 double norm_serial(int p, double* &gradient)

```

```

93 {
94     int i;
95     double norm=0;
96
97     for (i=0; i<p+1; i++)
98     {
99         norm += gradient[i]*gradient[i];
100     }
101
102     return sqrt(norm);
103
104 }
105
106 void gradient_descent_parallel(double* &X, double* &X_T, double* &y, double*
    &theta, double* &gradient)
107 {
108     double* X_theta;
109     int i;
110     double learning_rate=0.1, eps=0.0000000001;
111     X_theta = new double[n*1];
112     do
113     {
114
115         matmul_parallel(n, p+1, 1, X, theta, X_theta);
116         # pragma omp parallel shared (X_theta, n) private (i)
117         # pragma omp for
118         for (i=0; i<n; i++)
119             X_theta[i] -= y[i];
120
121         matmul_parallel(p+1, n, 1, X_T, X_theta, gradient);
122
123         # pragma omp parallel shared (gradient, p) private (i)
124         # pragma omp for
125         for (i=0; i<p+1; i++)
126             gradient[i] = gradient[i]*2/n;
127
128         # pragma omp parallel shared (gradient, p) private (i)
129         # pragma omp for
130         for (int i = 0; i<p+1; i++)
131             theta[i] = theta[i] - learning_rate*gradient[i];
132     }
133     while (norm_parallel(p, gradient)>eps);
134     delete[] X_theta;

```

```

135 }
136
137 void gradient_descent_serial(double* &X, double* &X_T, double* &y, double* &
    theta, double* &gradient)
138 {
139     double* X_theta;
140     int i;
141     double learning_rate=0.1, eps=0.0000000001;
142     X_theta = new double[n*1];
143     do
144     {
145         matmul_serial(n, p+1, 1, X, theta, X_theta);
146         for (i=0; i<n; i++)
147             X_theta[i] -= y[i];
148
149         matmul_serial(p+1, n, 1, X_T, X_theta, gradient);
150
151         for (i=0; i<p+1; i++)
152             gradient[i] = gradient[i]*2/n;
153
154         for (int i = 0; i<p+1; i++)
155             theta[i] = theta[i] - learning_rate*gradient[i];
156     }
157     while (norm_parallel(p, gradient)>eps);
158     delete[] X_theta;
159 }
160
161 int main()
162 {
163     double* X;
164     double* X_T;
165     double* y;
166     double* theta;
167     double* gradient;
168     read_data(X, X_T, y, theta, gradient);
169
170     double wtime;
171     wtime = omp_get_wtime ( );
172     gradient_descent_serial(X, X_T, y, theta, gradient);
173     wtime = omp_get_wtime ( ) - wtime;
174     printf ( "Thời gian tính toán tuần tự = %g\n", wtime );
175     printf("Kết quả tính toán tuần tự: \n");
176     for (int i=0; i<p+1; i++)

```

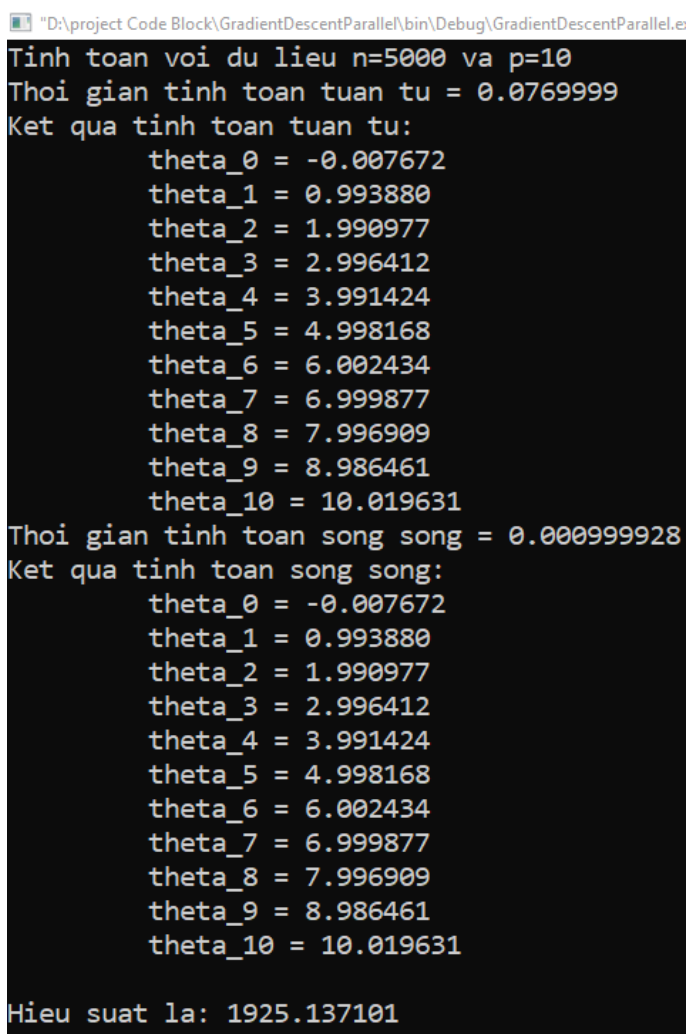
```
177     {
178         printf("\t theta_%d = %lf \n",i, theta[i]);
179     }
180
181     wtime = omp_get_wtime ( );
182     gradient_descent_parallel(X, X_T, y, theta, gradient);
183     wtime = omp_get_wtime ( ) - wtime;
184     printf ( "Thời gian tính toán song song = %g\n", wtime );
185     printf("Ket qua tính toán song song: \n");
186     for (int i=0; i<p+1; i++)
187     {
188         printf("\t theta_%d = %lf \n",i, theta[i]);
189     }
190     delete[] X;
191     delete[] X_T;
192     delete[] y;
193     delete[] theta;
194     delete[] gradient;
195 }
```

Mã 3.4: Toàn bộ chương trình

CHƯƠNG 4. ĐÁNH GIÁ KẾT QUẢ

4.1 Kiểm tra tính đúng đắn của chương trình

Như đã đề cập ở mục 3.1.1, tập dữ liệu được khởi tạo với $\theta_j = j$. Ta sẽ chạy thử chương trình với tập dữ liệu gồm 5000 quan sát của 10 biến độc lập để kiểm tra $\hat{\theta}$ thu được có gần đúng với θ thực tế không?



```
"D:\project Code Block\GradientDescentParallel\bin\Debug\GradientDescentParallel.e
Tinh toan voi du lieu n=5000 va p=10
Thoi gian tinh toan tuan tu = 0.0769999
Ket qua tinh toan tuan tu:
    theta_0 = -0.007672
    theta_1 = 0.993880
    theta_2 = 1.990977
    theta_3 = 2.996412
    theta_4 = 3.991424
    theta_5 = 4.998168
    theta_6 = 6.002434
    theta_7 = 6.999877
    theta_8 = 7.996909
    theta_9 = 8.986461
    theta_10 = 10.019631
Thoi gian tinh toan song song = 0.000999928
Ket qua tinh toan song song:
    theta_0 = -0.007672
    theta_1 = 0.993880
    theta_2 = 1.990977
    theta_3 = 2.996412
    theta_4 = 3.991424
    theta_5 = 4.998168
    theta_6 = 6.002434
    theta_7 = 6.999877
    theta_8 = 7.996909
    theta_9 = 8.986461
    theta_10 = 10.019631
Hieu suat la: 1925.137101
```

Hình 4.1. Kết quả chương trình khá chính xác với $n = 5000, p = 10$

4.2 So sánh chương trình tuần tự và song song

Tác giả đã thử chạy chương trình với số lượng luồng là 4 cùng dữ liệu có kích thước khác nhau và thu được kết quả:

Số lượng quan sát n	Số lượng biến p	Thời gian thực hiện song song	Thời gian thực hiện tuần tự	Hiệu suất
5000	1	0.00100017	0.026	649.892729
5000	10	0.000999928	0.0910001	2275.166905
5000	50	0.00199986	0.353	4412.818908
5000	100	0.00300002	0.788	6566.621632
5000	200	0.00600004	1.81	7541.614679
5000	300	0.00699997	3.021	10789.333277
5000	500	0.0120001	6.202	12920.742768

Bảng 4.1 So sánh chương trình tuần tự và song song

Như vậy, chương trình song song vượt trội hoàn toàn so với chương trình tuần tự.

CHƯƠNG 5. Kết LUẬN

Trong phạm vi nội dung của tiểu luận, một số kết quả đạt được của tiểu luận bao gồm:

- Thành công trong việc giới thiệu và tổng hợp các khái niệm cơ bản trong tối ưu, machine learning và tính toán song song.
- Ứng dụng được thuật toán Gradient Descent vào bài toán huấn luyện mô hình hồi quy tuyến tính.
- Cài đặt thành công chương trình tính toán song song sử dụng OpenMP và đạt kết quả vượt trội so với chương trình tuần tự.

Với kết quả thu được, tiểu luận đã chứng minh được vai trò quan trọng của tính toán song song trong bối cảnh khoa học kỹ thuật ngày càng phát triển với nhiều bài toán có khối lượng tính toán rất lớn. Ngoài ra, tiểu luận cũng đóng góp một phần nhỏ giúp cho cộng đồng tính toán song song ngày càng phát triển.

Với sự trình bày dễ hiểu và đơn giản, tiểu luận rất có tiềm năng để chỉnh sửa, thay đổi và cải tiến. Những hướng phát triển tiếp theo của tiểu luận:

- Thực thi chương trình trên mạng diện rộng.
- Vận dụng thêm các thuật toán tối ưu khác như Stochastic Gradient Descent, Momentum, AdaGrad, Adam, v.v...
- Áp dụng vào mô hình phức tạp hơn như Neurol Network,...

Tài liệu tham khảo

- [1] N. T. B. Kim, *Các Phương pháp Tối ưu, Lý thuyết và Thuật toán*. Nhà xuất bản Đại học Bách khoa Hà Nội, 2008.
- [2] A. Geron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 978-1-492-03264-9, O'reilly, 2nd ed., 2019.
- [3] L. X. Lý, “Bài giảng mô hình hồi quy tuyến tính,” 2020.
- [4] Đoàn Duy Trung, “Bài giảng lập trình song song với openmp,” 2021.