

ĐẠI HỌC HUẾ

KHOA KỸ THUẬT VÀ CÔNG NGHỆ





# BÁO CÁO

## ĐỒ ÁN (TIỂU LUẬN, BÀI TẬP LỚN)

**Học kỳ I, năm học 2021 - 2022**

**Học phần:**

### **HỌC MÁY 1**

***Đề tài: Gradient Descent***

**Số phách**

*(Do hội đồng chấm thi ghi)*

Thừa Thiên Huế, ngày 18 tháng 1 năm 2022



ĐẠI HỌC HUẾ

KHOA KỸ THUẬT VÀ CÔNG NGHỆ





# BÁO CÁO

## ĐỒ ÁN (TIỂU LUẬN, BÀI TẬP LỚN)

**Học kỳ I, năm học 2021 - 2022**

**Học phần:**

### HỌC MÁY 1

***Đề tài: Gradient Descent***

###### Giảng viên hướng dẫn: Lê Trung Hiếu

###### Sinh viên thực hiện: Nguyễn Trịnh Tấn Đạt

###### Mai Nhật Minh

###### Lê Thị Ngọc Thắm

###### Nguyễn Thành Lâm

###### Nguyễn Tiến Anh Quân

###### Lớp: Khoa học dữ liệu & Trí tuệ nhân tạo

**Số phách**

*(Do hội đồng chấm thi ghi)*

Thừa Thiên Huế, ngày 18 tháng 1 năm 2022

ĐẠI HỌC HUẾ

KHOA KỸ THUẬT VÀ CÔNG NGHỆ



PHIẾU ĐÁNH GIÁ ĐỒ ÁN/TIỂU LUẬN/BÀI TẬP LỚN

**Học kỳ I, năm học 2021 - 2022**

|  |  |
| --- | --- |
| **Cán bộ chấm thi 1** | **Cán bộ chấm thi 2** |
| **Nhận xét:**  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  **Điểm đánh giá của CBCT1:**  Bằng số: .........................................  Bằng chữ: ....................................... | **Nhận xét:**  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  ..............................................................  **Điểm đánh giá của CBCT2:**  Bằng số: .........................................  Bằng chữ: ....................................... |

Điểm kết luận: ...........................................................................................................

Bằng số:.....................................................................................................................

Bằng chữ: .................................................................................................................

*Thừa Thiên Huế, ngày tháng năm 2021*

|  |  |
| --- | --- |
| **Cán bộ chấm thi 1**  *(Ký và ghi rõ họ và tên)* | **Cán bộ chấm thi 2**  *(Ký và ghi rõ họ và tên)* |

**MỤC LỤC**

[CHƯƠNG I: Giới thiệu Gradient Descent 5](#_Toc93581942)

[I. Gradient Descent là gì 5](#_Toc93581943)

[II. Tại sao lại nên sử dụng Gradient Descent 5](#_Toc93581944)

[III. Cách tìm được trọng số tối ưu. 5](#_Toc93581945)

[CHƯƠNG II: Learning Rate 6](#_Toc93581946)

[CHƯƠNG III: Phân tích về Gradient Descent 7](#_Toc93581947)

[I. Ví dụ về Gradient Descent 7](#_Toc93581948)

[II. Gradient Descent và Momentum 12](#_Toc93581949)

[III. Nesterov accelerated gradient (NAG) 13](#_Toc93581950)

[IV. Các thuật toán khác 14](#_Toc93581951)

[V. Biến thể của Gradient Descent 14](#_Toc93581952)

[VI. Stopping Criteria (điều kiện dừng) 16](#_Toc93581953)

[VII. Một phương pháp tối ưu đơn giản khác: Newton’s method 16](#_Toc93581954)

[CHƯƠNG IV: THAM KHẢO 18](#_Toc93581955)

1. Giới thiệu Gradient Descent
   1. Gradient Descent là gì

Gradient Descent là một hình thức tối ưu học máy bằng cách lặp đi lặp lại để giảm đi Cost Function, nhằm đưa ra các mô hình dự đoán chính xác.

Cost Function (C) hoặc Lost Function đo lường sự khác biệt giữa sản lượng thực tế và sản lượng dự đoán từ mô hình. Hàm chi phí là một hàm lồi.

* 1. Tại sao lại nên sử dụng Gradient Descent

Trong mạng nơ-ron, mục tiêu của ta là tạo ra mô hình có trọng số (w) được tối ưu hóa để có thể đưa ra các dự đoán tốt hơn, và việc sử dụng Gradient Descent, sẽ giúp chúng ta sẽ nhận được một trọng số được tối ưu nhất.

* 1. Cách tìm được trọng số tối ưu.

Có thể giải thích vấn đề này dễ hiểu nhất bằng bài toán “mountain problem” cổ điển.

* Bài toán : Chúng ta cần tìm để đi đến điểm thấp nhất của dãy núi trong khi không hề có tầm nhìn và không hề biết bản thân đang ở đỉnh, ở giữa hay đã ở gần với điểm thấp nhất đó.
* Để giải quyết vấn đề, đơn giản nhất là ta có thể kiểm tra xung quanh và xác định điểm mà ta có thể đi xuống, bằng cách lặp đi lặp lại, và cứu kiểm tra đến lúc không còn điểm nào để đi xuống nữa, nghĩa là chúng ta đã chạm tới đáy.

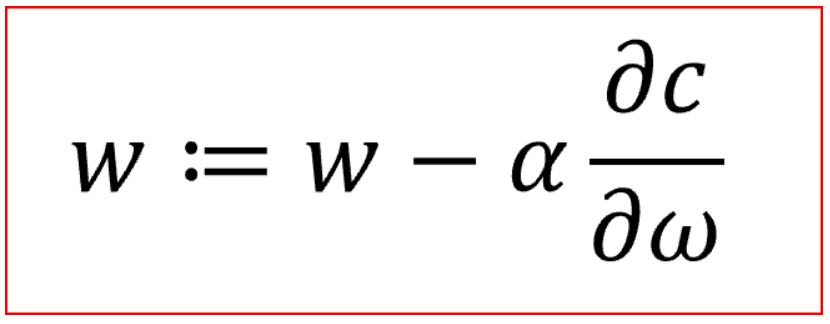
Gradient Descent cũng giống như bài toán trên, nhưng nó sẽ giải quyết vấn đề tương tự theo hướng toán học.

Chúng tôi khởi tạo ngẫu nhiên tất cả các trọng số của mạng nơ-ron thành một giá trị gần bằng 0 nhưng không bằng 0.

chúng tôi tính toán gradient, ∂c / ∂ω là đạo hàm riêng của mất mát liên quan đến trọng số.

α là Learning Rate, giúp điều chỉnh trọng số liên quan đến độ dốc xuống.

Công thức:



Chúng ta cần cập nhật đồng thời trọng số cho tất cả các nơ-ron.

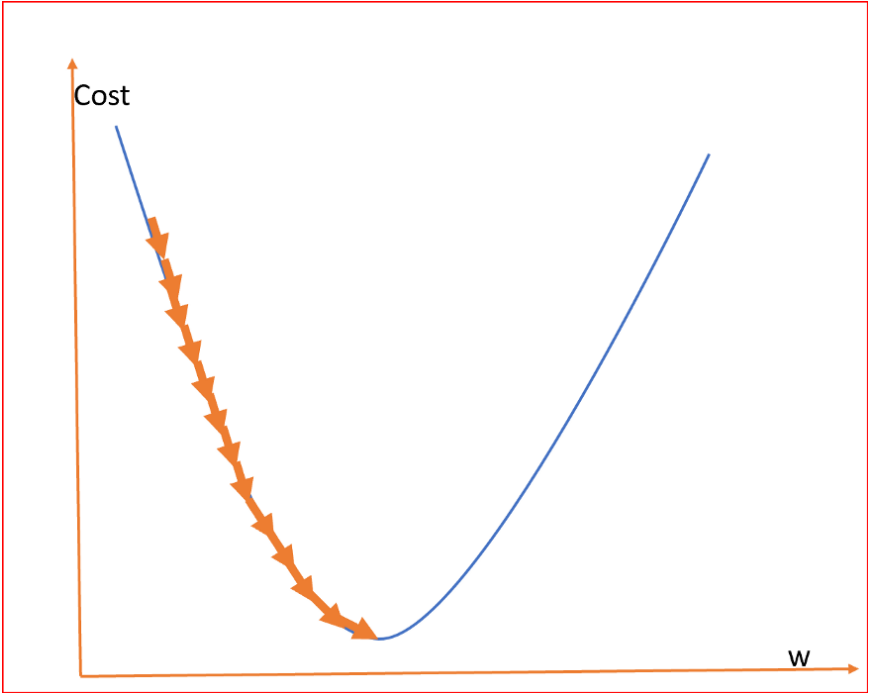
1. Learning Rate

Learning Rate là một siêu tham số sử dụng trong việc huấn luyện các mạng nơ ron. Giá trị của nó là một số dương, thường nằm trong khoảng giữa 0 và 1. Learning Rate kiểm soát tốc độ mô hình thay đổi các trọng số để phù hợp với bài toán. Tốc độ học lớn giúp mạng nơ ron được huấn luyện nhanh hơn nhưng cũng có thể làm giảm độ chính xác.

Learning Rate thường được khởi tạo ngẫu nhiên.

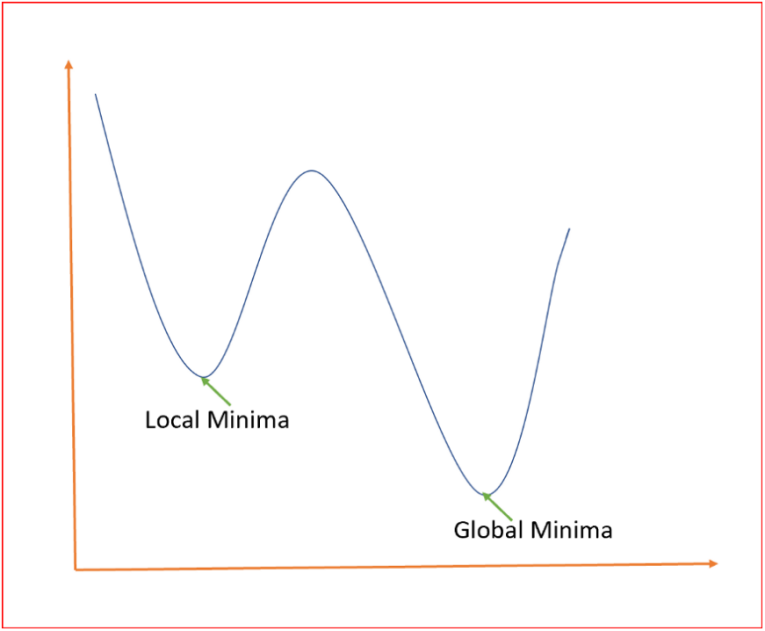
Giá trị Learning Rate quá cao sẽ không cho phép Gradient tụ lại.

Mục tiêu của chúng ta là giảm thiểu cost function để tìm được giá trị tối ưu nhất cho trọng số. Sau khi lặp lại nhiều lần với các giá trị trọng số khác nhau, ta tính được lượng mất mát tối thiểu như hình dưới đây:



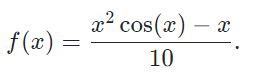
Có thể có “hai đáy khác nhau của ngọn núi” theo cách tương tự, chúng ta cũng có thể lấy điểm tối thiểu cục bộ (local minimum) và toàn diện ( global minimum ) giữa mất mát và trọng số.

Global Minima là điểm thấp nhất của cả miền, còn Local Minima là ta nhận được một điểm tương đối tối thiểu nhưng không phải là điểm tối thiểu chung như hình dưới đây.

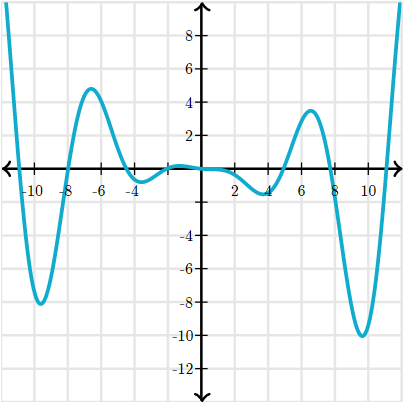


1. Phân tích về Gradient Descent
   1. Ví dụ về Gradient Descent

Cho f(x):

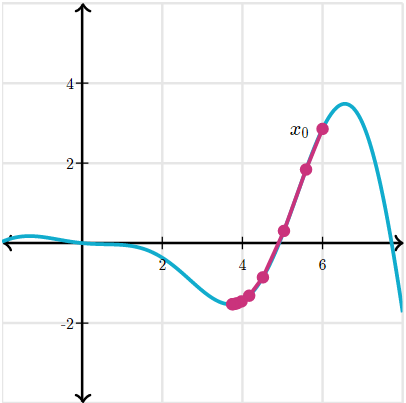


Ta có đồ thị:

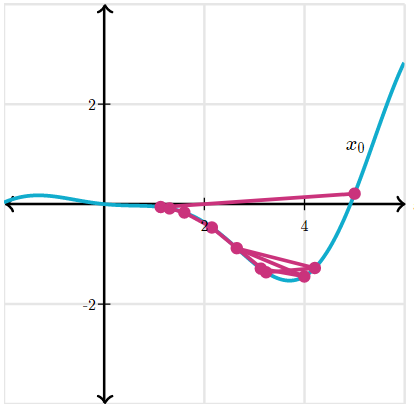


Ta có thể thấy rất nhiều Local Minima, tùy vào điều chỉnh của ta mà Gradient Descent sẽ tìm đến những cái khác nhau.

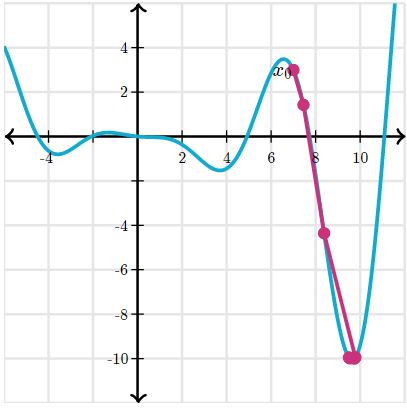
Nếu ta chọn x = 6 và α = 0.2 thì sau 10 bước ta sẽ tìm được minimum gần bằng điểm x = 4.



Nếu cho x=6 như cũ, nhưng tăng α = 1.5, nó chạm đến hạn chế của thuật toán nên thành ra có số bước quá lớn để gradient Descent có thể tìm được minimum.

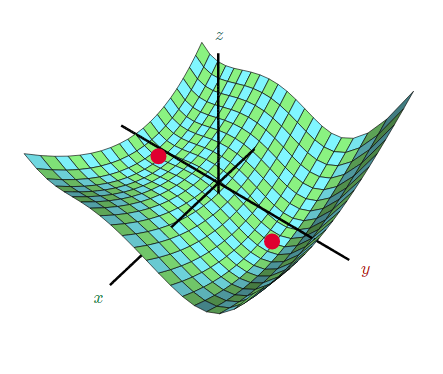


Nếu ta cho x=7 và α = 0,2 ta sẽ tìm ra được điểm minimum hoàn toàn khác.

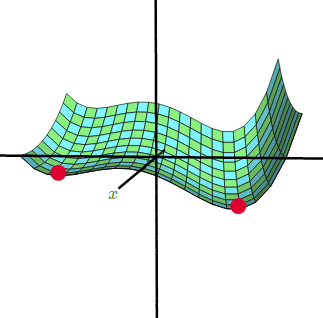


Qua ví dụ trên ta cũng thấy được một vài điểm thiếu sót của Gradient Descent ví dụ như:

* Gradient Descent chỉ có thể tìm ra được Local Minima, vì một khi nó tìm được Local Minima, nó không thể thoát ra để tìm được các điểm khác nữa cho tới khi kích thước bước lớn hơn kích thước của rãnh.



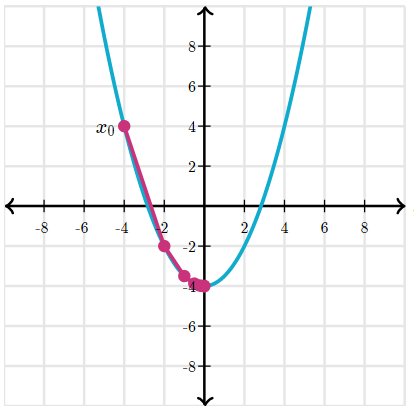
Ví dụ như ở hình trên, ở mỗi rảnh thì chúng ta lại có một điểm cực tiểu riêng, khi mà thuật toán Gradient chỉ cố đi xuống, và nó tìm được điểm cực tiểu, nơi mà các vùng xung quanh nó đều hướng lên, thì nó sẽ dừng lại.



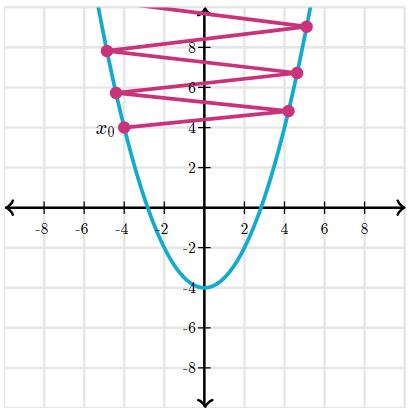
Nhìn từ góc độ này thì chúng ta thấy là có tận 2 Local Manima, và có một cái thấp hơn cái còn lại, nhưng không thể nào có chuyện Gradient lại phân biệt được đâu là Local đâu là Global Manima.

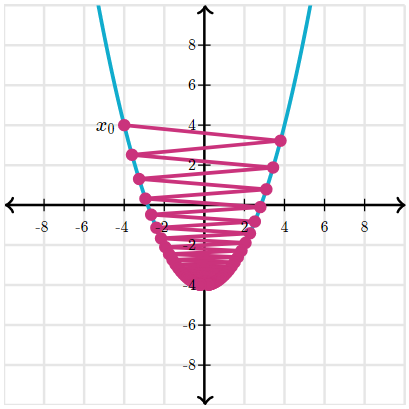
Hạn chế thứ hai là step size α, nếu α chúng ta chọn chuẩn xác, thì nó sẽ đi đến điểm cần tìm nhanh chóng và từng bước đều có hiệu quả cao

Ví dụ về chọn chuẩn xác bước α:

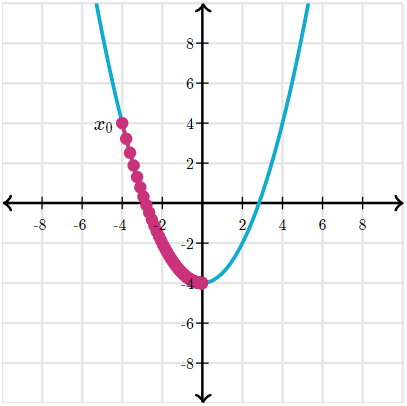


Nếu chọn bước quá lớn thì nó có khả năng sẽ không tìm được điểm minima, hoặc nếu may mắn nó vẫn chạy, thì nó sẽ tìm được điểm minima với rất nhiều bước.

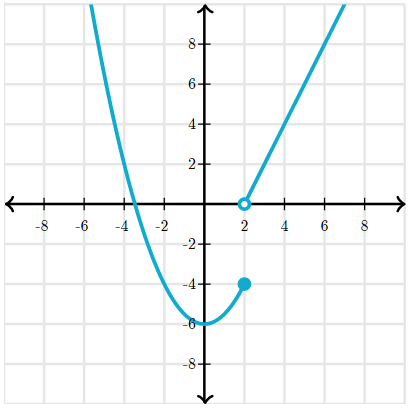




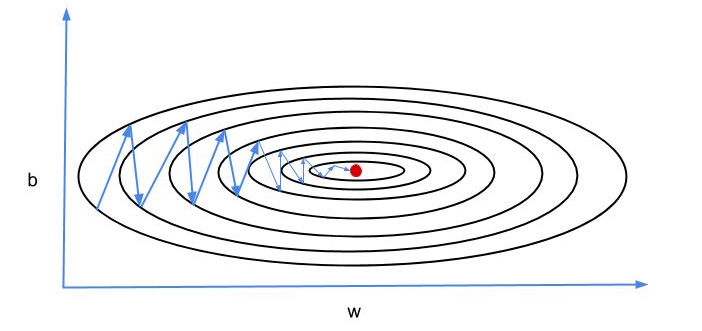
Nếu bước quá nhỏ thì sẽ tìm nhưng rất rất chậm



Một giới hạn thứ 3 của Gradient Descent đó chính là nó sẽ không thể thực hiện được trừ khi biểu đồ đó liên tục và có kết quả qua các bước khác nhau. Ví dụ về một dạng mà Gradient Descent không chạy được.



* 1. Gradient Descent và Momentum



Một trong những vấn đề với sự giảm dần gradient đơn giản là nó dao động quá nhiều trên trục tung (như thể hiện trong hình trên). Dao động này làm chậm sự hội tụ đến mức cực tiểu (chấm đỏ).

Những gì chúng ta muốn là ít dao động hơn trên trục tung và di chuyển nhanh hơn trên trục hoành về phía cực tiểu. Do đó, điều này sẽ dẫn đến việc học tập nhanh hơn và hội tụ ở mức tối thiểu của cảnh quan năng lượng.

Làm thế nào để chúng ta đạt được mục tiêu này? Đổ dốc theo momentum là một trong nhiều cách khả thi để làm điều đó. Gradient descent với momentum là một thuật toán tối ưu hóa dựa trên các yếu tố trọng số trung bình theo cấp số nhân - được áp dụng (giảm dần theo cấp số nhân) cho các điểm dữ liệu cũ hơn.

Đây là cách chúng tôi triển khai giảm độ dốc với động lượng:

Trên mỗi lần lặp lại giảm độ dốc theo lô hoặc lô nhỏ, trước tiên hãy tính toán các dẫn xuất, dW và dbvà sau đó nhận được mức trung bình có trọng số theo cấp số nhân như sau:

**vdW= βvdW+ ( 1 - β) dW**

**vdb= βvdb+ ( 1 - β) db**

Điều này sẽ thêm một siêu tham số khác β để học thuật toán. Thông thường, giá trị 0,9 được sử dụng cho β để lấy trung bình của 10 lần lặp lại gradient gần đây nhất. Không cần hiệu chỉnh độ lệch vì độ lệch sẽ biến mất sau vài lần lặp lại quá trình giảm độ dốc. Trong các phương trình 1 và 2 ở trên, các đạo hàm biểu thị gia tốc trong khi các số hạng động lượng là vận tốc. β thuật ngữ có thể được coi là ma sát.

Bây giờ chúng ta có thể cập nhật trọng số bằng cách sử dụng trung bình có trọng số theo cấp số nhân (hơn so với các phái sinh thông thường);

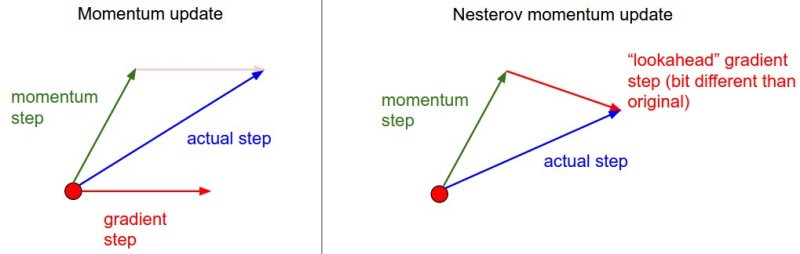
W: = W- αVdW

b : = b - αvdb

Điều này sẽ làm mịn quá trình giảm dần độ dốc và các dao động dọc sẽ trung bình bằng không. Như một hệ quả, dốc xuống sẽ hội tụ nhanh chóng vì nó không thực hiện từng bước độc lập với những bước trước đó. Tốt hơn là thực hiện các bước trước đó để đạt được động lực từ chúng.

* 1. Nesterov accelerated gradient (NAG)

Một hạn chế chúng ta có thể thấy là : Khi tới gần đích, momemtum vẫn mất khá nhiều thời gian trước khi dừng lại. Lý do lại cũng chính là vì có đà. Có một phương pháp khác tiếp tục giúp khắc phục điều này, phương pháp đó mang tên Nesterov accelerated gradient (NAG), giúp cho thuật toán hội tụ nhanh hơn.



NAG là một biến thể của trình tối ưu hóa momentum. Nó đo lường độ dốc của cost function đi trước một chút so với hướng của momentum thay vì ở vị trí cục bộ.

Sự khác biệt nhỏ này cho phép tối ưu hóa nhanh hơn vì nói chung, vectơ momentum sẽ hướng tới mức tối ưu. Do đó, mỗi bước chính xác hơn một chút so với tối ưu hóa momentum và cải tiến nhỏ đó sẽ tăng lên theo thời gian. Ngoài ra, dao động bị giảm với NAG bởi vì khi xung lượng đẩy quả nặng qua điểm tối ưu, thì gradient phía trước sẽ đẩy nó trở lại điểm tối ưu.

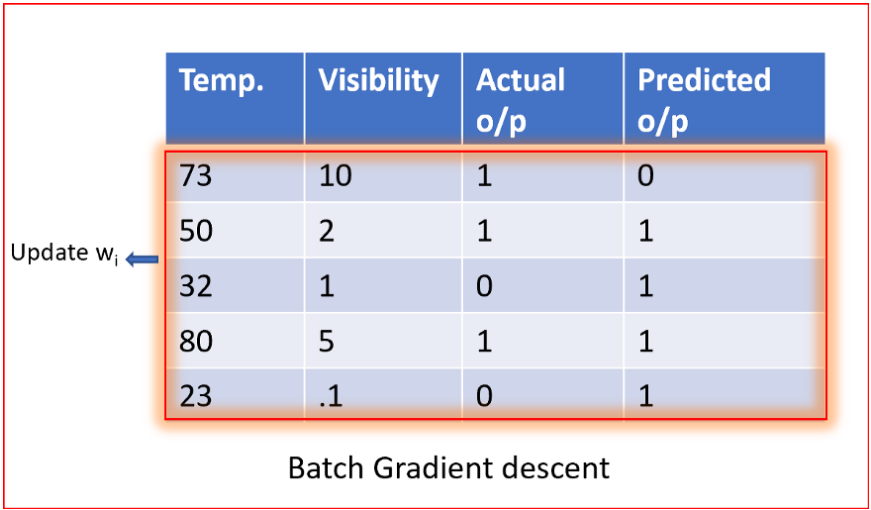
* 1. Các thuật toán khác

Ngoài hai thuật toán trên, có rất nhiều thuật toán nâng cao khác được sử dụng trong các bài toán thực tế, đặc biệt là các bài toán Deep Learning. Có thể nêu một vài từ khóa như Adagrad, Adam, RMSprop,…

* 1. Biến thể của Gradient Descent
     1. Batch Gradient Descent

Trong Batch Gradient Descent chúng ta sẽ xử dụng cả 1 dataset để tính toán các yếu tố ví dụ như đoạn dốc của cost function cho mỗi lần lặp và cập nhật trọng số. Đổi lại bằng việc dùng cả Dataset để tính toán, thì nó sẽ làm chậm lại phép gradient.

Nếu dataset là rất lớn, hàng triệu thậm chí hàng tỉ điểm, nó sẽ được lưu trữ lại nếu như đó là tính toán chuyên sâu.



Điểm lợi của Batch Gradient Descent:

* Các phép phân tích trọng số lẫn tỷ lệ hội tụ đều rất dễ hiểu.
* Điểm bất lợi của Batch Gradient Descent:
* Nó sẽ thực hiện lại các phép tính thừa nếu như dataset là quá lớn.
* Nó sẽ rất chậm và sẽ khó kiểm soát với các dataset lớn vì vấn đề bộ nhớ.
* Có một số cách tính phức tạp nhưng có thể hiệu quả hơn việc tính toán bằng cả một dataset lớn.
  + 1. Stochastic Gradient Descent.

Trong stochastic gradient descent, chúng ta sử dụng một điểm dữ liệu hoặc ví dụ duy nhất để tính toán độ dốc và cập nhật trọng số với mỗi lần lặp lại.

  Nó có thể được coi là xấp xỉ ngẫu nhiên của tối ưu hóa giảm dần gradient , vì nó thay thế gradient thực tế (được tính từ toàn bộ tập dữ liệu ) bằng một ước tính của nó (được tính từ một tập hợp con được chọn ngẫu nhiên của dữ liệu). Đặc biệt trong các bài toán tối ưu hóa chiều cao, điều này làm giảm gánh nặng tính toán, đạt được số lần lặp lại nhanh hơn trong giao dịch để có tỷ lệ hội tụ thấp hơn.

Có một nhược điểm của bản chất Stochastic của SGD, tức là một khi nó đạt đến gần giá trị tối thiểu thì nó không lắng xuống, thay vào đó nó sẽ nảy lên xung quanh, điều này mang lại cho chúng ta một giá trị tốt cho các tham số mô hình nhưng không tối ưu. Có thể giải quyết bằng cách giảm học tỷ giá ở mỗi bước có thể làm giảm độ nảy và SGD có thể giảm xuống ở mức tối thiểu toàn cầu sau một thời gian.

* + 1. Mini Batch Gradient descent

Là một biến thể của Stochastic gradient descent nhưng thay vì dùng 1 mẫu ngẫu nhiên, ta sẽ dùng 1 tập nhỏ, và dĩ nhiên nó được sử dụng rộng rãi vì hội tụ nhanh hơn và ổn định hơn.

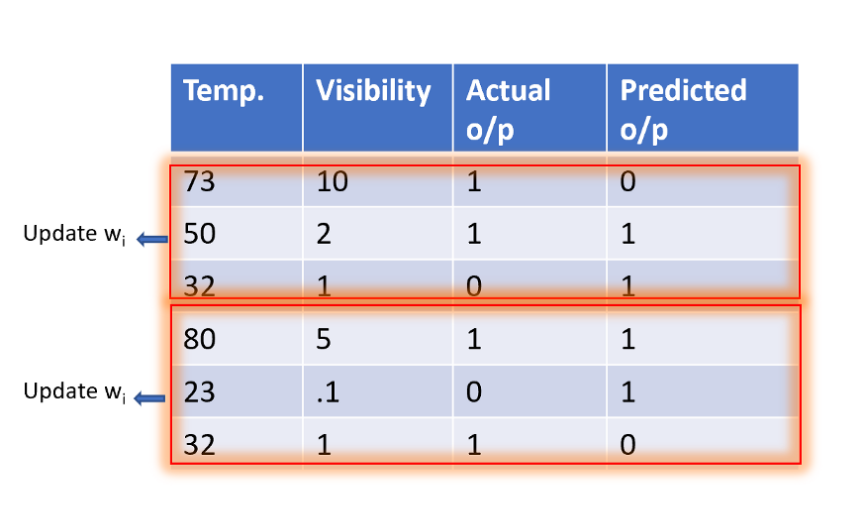
Khi dùng các mẫu nhỏ khác nhau, thì nó sẽ bỏ đi những thứ tạp nham ( phương sai của trọng số là 1 ví dụ ) từ đó làm mọi thứ ổn định và nhanh chóng hơn.

Lợi ích của việc dùng mini-batch:

* Giảm đi các sai số của việc cập nhật tham số, làm cho việc hội tụ dễ dàng hơn
* Học tập nhanh hơn.
* Hữu dụng cho việc ước tính địa điểm của điểm minimum

Nhược điểm:

* Mất mát ( loss ) được tính qua từng cụm mini-batch một, nên để tính tổng mất mát thì phải tính tổng của toàn bộ trong các cụm mini-batch.



* 1. Stopping Criteria (điều kiện dừng)

Trả lời cho câu hỏi khi nào thì thuật toán đã hội tụ và dừng lại ?

Trong thực nghiệm, có một vài phương pháp như dưới đây:

Giới hạn số vòng lặp: đây là phương pháp phổ biến nhất và cũng để đảm bảo rằng chương trình chạy không quá lâu. Tuy nhiên, một nhược điểm của cách làm này là có thể thuật toán dừng lại trước khi đủ gần với nghiệm.

So sánh gradient của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại. Phương pháp này cũng có một nhược điểm lớn là việc tính đạo hàm đôi khi trở nên quá phức tạp (ví dụ như khi có quá nhiều dữ liệu), nếu áp dụng phương pháp này thì coi như ta không được lợi khi sử dụng SGD và mini-batch GD.

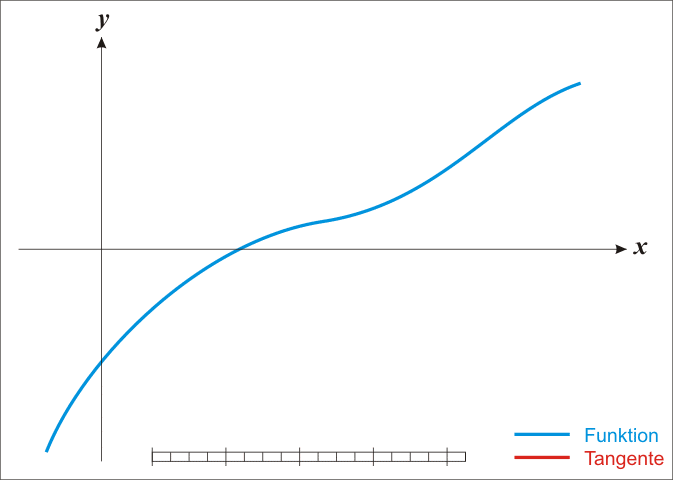
So sánh giá trị của hàm mất mát của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại. Nhược điểm của phương pháp này là nếu tại một thời điểm, đồ thị hàm số có dạng bẳng phẳng tại một khu vực nhưng khu vực đó không chứa điểm local minimum (khu vực này thường được gọi là saddle points), thuật toán cũng dừng lại trước khi đạt giá trị mong muốn.

Trong SGD và mini-batch GD, cách thường dùng là so sánh nghiệm sau một vài lần cập nhật. Trong đoạn code Python phía trên về SGD, tôi áp dụng việc so sánh này mỗi khi nghiệm được cập nhật 10 lần. Việc làm này cũng tỏ ra khá hiệu quả.

* 1. Một phương pháp tối ưu đơn giản khác: Newton’s method

**Newton’s method cho giải phương trình f(x)=0f(x)=0**

Thuật toán Newton’s method được mô tả trong hình động minh họa dưới đây:



Ý tưởng giải bài toán **f(x)=0f(x)=0** bằng phương pháp Newton’s method như sau. Xuất phát từ một điểm x0x0 được cho là gần với nghiệm x∗x∗. Sau đó vẽ đường tiếp tuyến (mặt tiếp tuyến trong không gian nhiều chiều) với đồ thị hàm số **y=f(x)y=f(x)** tại điểm trên đồ thị có hoành độ x0x0.

Giao điểm x1x1 của đường tiếp tuyến này với trục hoành được xem là gần với nghiệm x∗x∗ hơn. Thuật toán lặp lại với điểm mới x1x1 và cứ như vậy đến khi ta được :

**f(xt)≈0f(xt)≈0.**

Đó là ý nghĩa hình học của Newton’s method, chúng ta cần một công thức để có thể dựa vào đó để lập trình. Việc này không quá phức tạp với các bạn thi đại học môn toán ở VN. Thật vậy, phương trình tiếp tuyến với đồ thị của hàm f(x)f(x) tại điểm có hoành độ xtxt là:

**y=f′(xt)(x−xt)+f(xt)y=f′(xt)(x−xt)+f(xt)**.

Giao điểm của đường thẳng này với trục xx tìm được bằng cách giải phương trình vế phải của biểu thức trên bằng 0, tức là:

**x=xt−f(xt)f′(xt)≜xt+1x=xt−f(xt)f′(xt)≜xt+1**

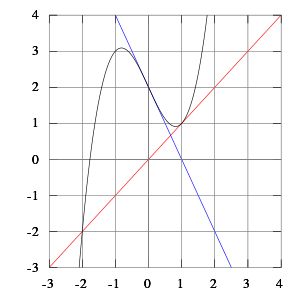
Newton’s method trong bài toán tìm local minimun

Áp dụng phương pháp này cho việc giải phương trình f′(x)=0f′(x)=0 ta có:xt+1=xt−(f”(xt))−1f′(xt)xt+1=xt−(f”(xt))−1f′(xt)

Và trong không gian nhiều chiều với θθ là biến:θ=θ−H(J(θ))−1∇θJ(θ)θ=θ−H(J(θ))−1∇θJ(θ)trong đó H(J(θ))H(J(θ)) là đạo hàm bậc hai của hàm mất mất (còn gọi là [Hessian matrix](https://en.wikipedia.org/wiki/Hessian_matrix)). Biểu thức này là một ma trận nếu θθ là một vector. Và H(J(θ))−1H(J(θ))−1 chính là nghịch đảo của ma trận đó.

Hạn chế của Newton’s method

Điểm khởi tạo phải rất gần với nghiệm x∗x∗. Ý tưởng sâu xa hơn của Newton’s method là dựa trên khai triển Taylor của hàm số f(x)f(x) tới đạo hàm thứ nhất:0=f(x∗)≈f(xt)+f′(xt)(xt−x∗)0=f(x∗)≈f(xt)+f′(xt)(xt−x∗)Từ đó suy ra: x∗≈xt−f(xt)f′(xt)x∗≈xt−f(xt)f′(xt). Một điểm rất quan trọng, khai triển Taylor chỉ đúng nếu xtxt rất gần với x∗x∗! Dưới đây là một ví dụ kinh điển trên Wikipedia về việc Newton’s method cho một dãy số phân kỳ (divergence).

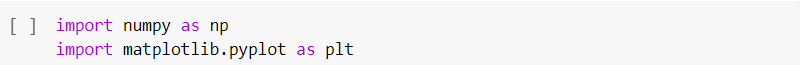


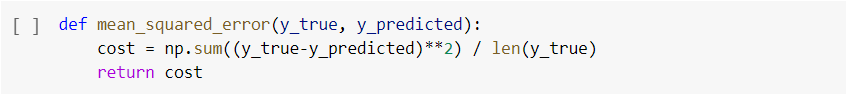
Nhận thấy rằng trong việc giải phương trình f(x)=0f(x)=0, chúng ta có đạo hàm ở mẫu số. Khi đạo hàm này gần với 0, ta sẽ được một đường thằng song song hoặc gần song song với trục hoành. Ta sẽ hoặc không tìm được giao điểm, hoặc được một giao điểm ở vô cùng. Đặc biệt, khi nghiệm chính là điểm có đạo hàm bằng 0, thuật toán gần như sẽ không tìm được nghiệm!

Khi áp dụng Newton’s method cho bài toán tối ưu trong không gian nhiều chiều, chúng ta cần tính nghịch đảo của Hessian matrix. Khi số chiều và số điểm dữ liệu lớn, đạo hàm bậc hai của hàm mất mát sẽ là một ma trận rất lớn, ảnh hưởng tới cả memory và tốc độ tính toán của hệ thống.

1. THỰC HÀNH

Nhập thư viện

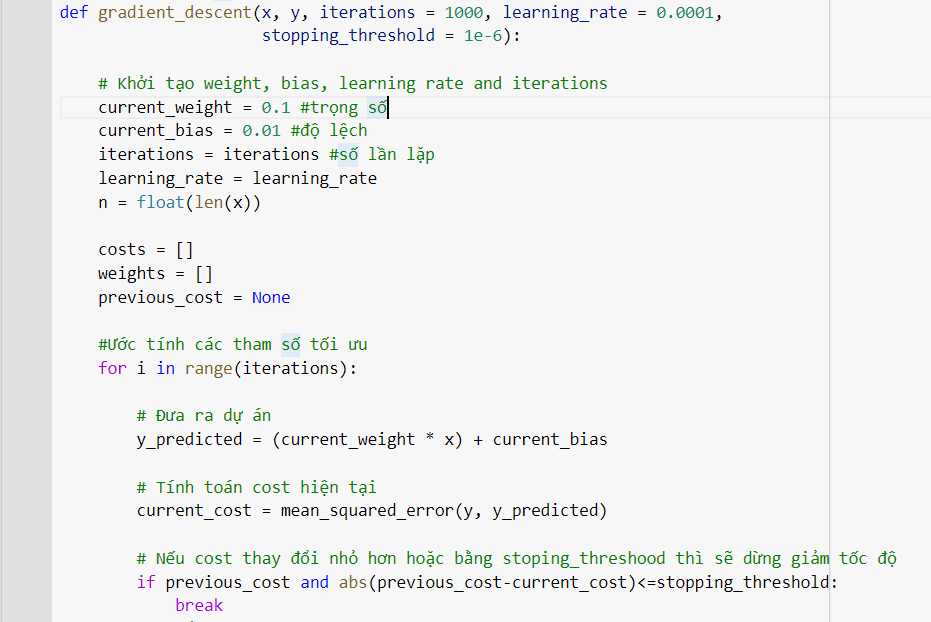


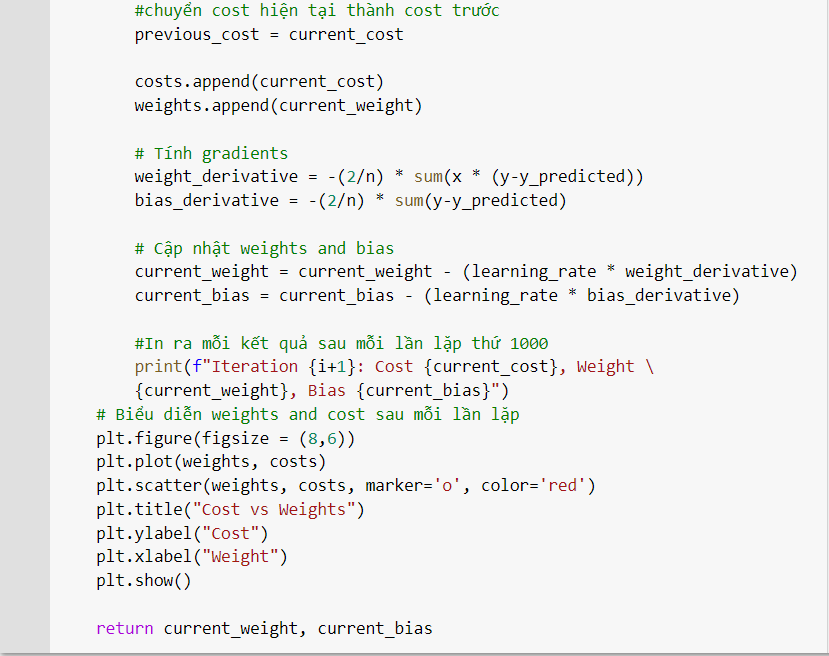
Lập công thức hàm loss và cost

Hàm Gradient Descent

Tạo iterations, learning\_rate, stopping\_threshold

Các siêu tham số có thể được điều chỉnh





Nhập dữ liệu



Source code:

<https://github.com/tandat-1305/Gradient_Descent/blob/main/Gradient_Descent.ipynb>

1. THAM KHẢO

[https://www.howkteam.vn/course/machine-learning-co-ban-voi-numpy/thuat-toan-gradient-descent-cho-linear-regression-3997?fbclid=IwAR1z\_ePxnHwD1ul53ia1rjoN66nJeF\_QU-k18\_\_uVcLONDL3vvJz-T6p2Wo](https://topdev.vn/blog/thuat-toan-gradient-descent/?fbclid=IwAR06QBYhlXFCVFMWVP5WHaJ8lOuZqC51JTlZhLkLmcxz54ew517IL1914po" \l "gradient-descent-la-gi)

[https://topdev.vn/blog/thuat-toan-gradient-descent/?fbclid=IwAR06QBYhlXFCVFMWVP5WHaJ8lOuZqC51JTlZhLkLmcxz54ew517IL1914po#gradient-descent-la-gi](https://topdev.vn/blog/thuat-toan-gradient-descent/?fbclid=IwAR06QBYhlXFCVFMWVP5WHaJ8lOuZqC51JTlZhLkLmcxz54ew517IL1914po" \l "gradient-descent-la-gi)

[https://l.facebook.com/l.php?u=https%3A%2F%2Fmachinelearningcoban.com%2F2017%2F01%2F12%2Fgradientdescent%2F%3Ffbclid%3DIwAR17PlHPMmBJmpB1s65rl2-EWAdHsKP2DvkIut6RAYlENKWqmBRuvwGx4xA&h=AT1uWEQ5NVlj2TN1lDtkhBHFRBzihda3eGOTN\_wQ2U4fiuq9MbiORXvbMsEE6b\_fg1mR8BHuzNjPQwe4mE9qqJEmkelFX5W-TqoFoBhgurPV93qsxjfdZABaBCa78nvbLTfYBsc\_cc5100s](https://topdev.vn/blog/thuat-toan-gradient-descent/?fbclid=IwAR06QBYhlXFCVFMWVP5WHaJ8lOuZqC51JTlZhLkLmcxz54ew517IL1914po" \l "gradient-descent-la-gi)