**Gradient Decent**

**Giới Thiệu:**

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. Ví dụ như các hàm mất mát trong hai bài Linear Regression và K-means Clustering. 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.



*Hình 1: biểu diễn các điểm local và global*

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 (đoạn này nghe rất quen thuộc, đúng không?). 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 – Đạo hàm ngược (viết gọn là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

**Gradient Decent cho hàm một biến:**

****

*Hình 2: parapol của hàm f(x)*

Giả sử là điểm ta tìm được sau vòng lặp **t**. Ta cần tìm một thuật toán để đưa về càng gần càng tốt.

Từ hình trên ta thấy:

Nếu đạo hàm của hàm số tại thì nằm về bên phải so với (và ngược lại). Để điểm tiếp theo gần với hơn, chúng ta cần di chuyển về phía bên trái, tức về phía âm. Nói cách khác, ta cần di chuyển ngược dấu với đạo hàm

Trong đó: là một đại lượng ngược dấu với đạo hàm

* càng xa về phía bên phải thì càng lớn hơn 0 (và ngược lại). Vậy, lượng di chuyển , một cách trực quan nhất, là tỉ lệ thuận với

Từ 2 nhận xét trên cho ta một cách cập nhật đơn giản:

Trong đó: (đọc là eta) là một số dương được gọi là learning rate (tốc độ học). Dấu trừ thể hiện việc chúng ta phải đi ngược với đạo hàm (Đây cũng chính là lý do phương pháp này được gọi là Gradient Descent - descent nghĩa là đi ngược). Các quan sát đơn giản phía trên, mặc dù không phải đúng cho tất cả các bài toán, là nền tảng cho rất nhiều phương pháp tối ưu nói chung và thuật toán Machine Learning nói riêng.

**Ví dụ đơn giản với Python:**

Xét hàm số với đạo hàm  **2x + 5cos(x).** Giả sử bắt đầu từ một điểm nào đó, tại vòng lặp thứ **,** chúng ta sẽ cập nhật như sau:

Các hàm số:

* **Grad\_Fx** để tính đạo hàm
* **Cost** để tính giá trị của hàm số. Hàm này không sử dụng trong thuật toán nhưng thường được dùng để kiểm tra việc tính đạo hàm của đúng không hoặc để xem giá trị của hàm số có giảm theo mỗi vòng lặp hay không.
* Hàm **Gradient\_Decent** là phần chính thực hiện thuật toán Gradient Desent nêu phía trên. Đầu vào của hàm số này là learning rate () và điểm bắt đầu. Thuật toán dừng lại khi đạo hàm có độ lớn đủ nhỏ

Các điểm khởi tạo khác nhau là và

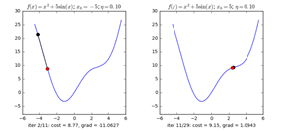
**import** numpy **as** np  
  
**def** grad\_Fx(x):  
 **return** 2\*x+ 5\*np.cos(x)**def** cost(x):  
 **return** x\*\*2 + 5\*np.sin(x)  
**def** Gradient\_Decent(eta, x0):  
 x = [x0]  
 **for** it **in** range(100):  
 x\_new = x[-1] - eta\*grad\_Fx(x[-1])  
 **if** abs(grad\_Fx(x\_new)) < 1e-3: *#Thuật toán dừng lại khi đạo hàm có độ lớn đủ nhỏ (0.001)* **break** x.append(x\_new)  
 **return** (x, it)  
(x1, it1) = Gradient\_Decent(.1, -5) *# đầu vào, learning rate = 0.1, điểm bắt đầu bằng -5*(x2, it2) = Gradient\_Decent(.1, 5) *# đầu vào, learning rate = 0.1, điểm bắt đầu bằng 5*print(**'với x1 = %f, cost = %f, hội tụ sau %d bước lặp'**%(x1[-1], cost(x1[-1]), it1))  
print(**'với x2 = %f, cost = %f, hội tụ sau %d bước lặp'**%(x2[-1], cost(x2[-1]), it2))

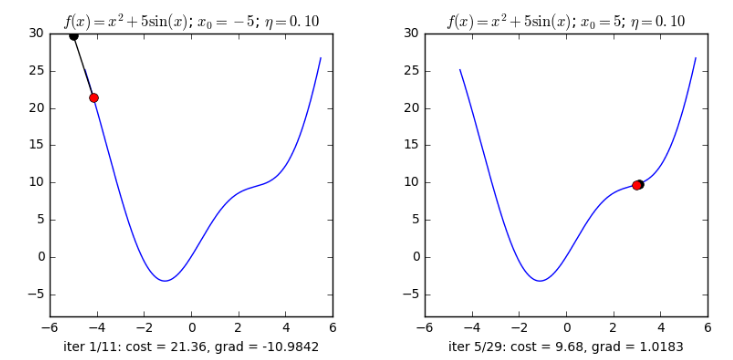
*Kết quả:*

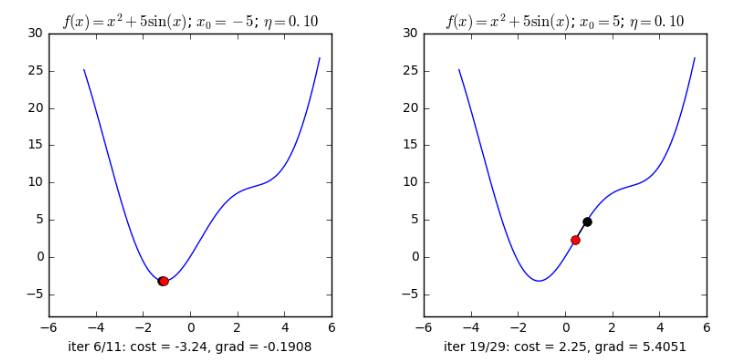
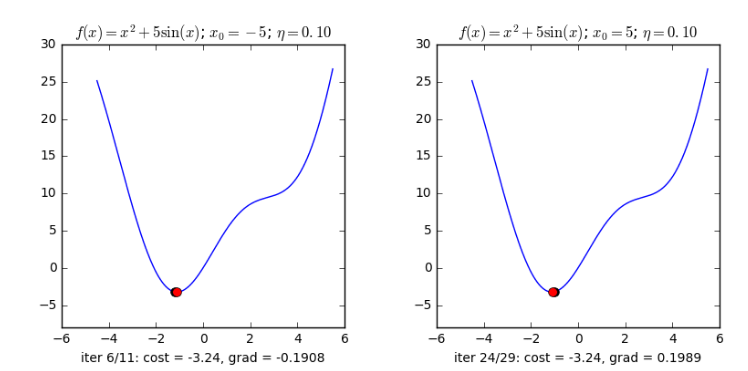
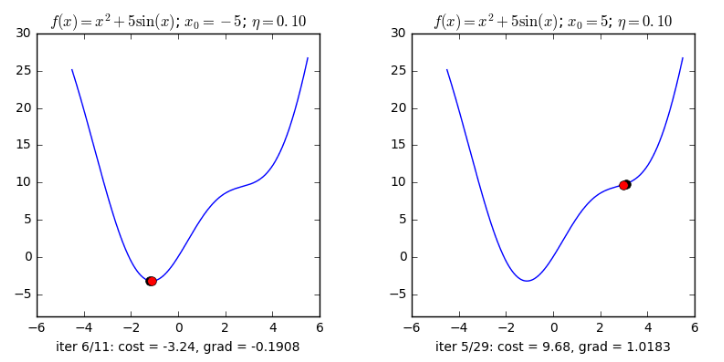
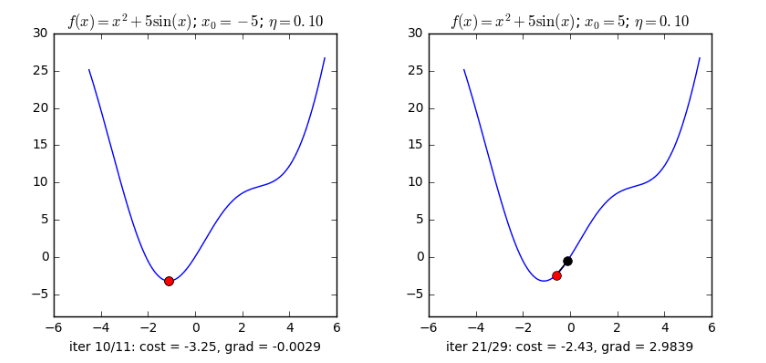
với x1 = -1.110667, cost = -3.246394, hội tụ sau 11 bước lặp

với x2 = -1.110341, cost = -3.246394, hội tụ sau 29 bước lặp

Vậy là với các điểm ban đầu khác nhau, thuật toán của chúng ta tìm được nghiệm gần giống nhau, mặc dù với tốc độ hội tụ khác nhau. Dưới đây là hình ảnh minh họa thuật toán GD cho bài toán này.





*** Hình 1: Hình 2:*

*Hình 3: Hình 4:*

*Hình 5: Hình 6:*

Từ hình minh họa trên ta thấy rằng ở hình bên trái, tương ứng với , nghiệm hội tụ nhanh hơn, vì điểm ban đầu gần với nghiệm hơn. Hơn nữa, với ở hình bên phải, đường đi của nghiệm có chứa một khu vực có đạo hàm khá nhỏ gần điểm có hoành độ bằng 2. Điều này khiến cho thuật toán la cà ở đây khá lâu. Khi vượt qua được điểm này thì mọi việc diễn ra rất tốt đẹp.

Nhận xét:

* Với các điểm ban đầu khác nhau, thuật toán tìm được nghiệm gần giống nhau, mặc dù với tốc độ hội tụ khác nhau.
* Tốc độ hội tụ của GD không những phụ thuộc vào điểm khởi tạo ban đầu mà còn phụ thuộc vào learning rate.

Việc lựa chọn learning rate rất quan trọng trong các bài toán thực tế. Việc lựa chọn giá trị này phụ thuộc nhiều vào từng bài toán và phải làm một vài thí nghiệm để chọn ra giá trị tốt nhất. Ngoài ra, tùy vào một số bài toán, GD có thể làm việc hiệu quả hơn bằng cách chọn ra learning rate phù hợp hoặc chọn learning rate khác nhau ở mỗi vòng lặp

**Gradient Decent cho hàm nhiều biến:**

Giả sử ta cần tìm global minimum cho hàm trong đó **θ** (theta) là một vector, thường được dùng để ký hiệu tập hợp các tham số của một mô hình cần tối ưu (trong Linear Regression thì các tham số chính là hệ số). Đạo hàm của hàm số đó tại một điểm **θ** bất kỳ được ký hiệu là (hình tam giác ngược đọc là *nabla*). Tương tự như hàm 1 biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán **,** sau đó, ở vòng lặp thứ , quy tắc cập nhật là: