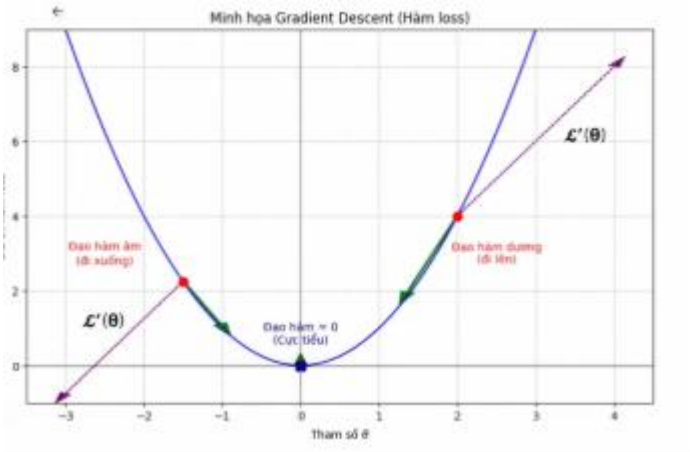
Các thành phần cơ bản trong mô hình học máy:

- X: Dữ liệu đầu vào (features)
    - Là thông tin thực tế (ví dụ: diện tích, số phòng ngủ,...)
- θ (theta): Tham số của mô hình
    - Là thứ mô hình cần học để đưa ra dự đoán tốt nhất
- y^ (y hat): Giá trị dự đoán
    - Mô hình dựa vào x và θ để tạo ra dự đoán y^

Làm cho dự đoán y^ gần với y nhất có thể. Để làm được điều này, ta cần:

- Định nghĩa một hàm loss (hàm mất mát):
    - L(θ;x, y) Đo lường độ lệch giữa y^ và y
- Tối ưu hóa tham số θ sao cho:
    - L(θ;x, y) càng nhỏ càng tốt
- Quá trình này gọi là tối ưu hóa thường dùng Gradient Descent

Mục tiêu: Tối thiểu hóa loss function bằng cách cập nhật tham số theo hướng giảm dần của đạo hàm



Giả sử đồ thị hàm **loss** có dạng parabol với điểm cực tiểu như hình. Xét điểm **θ** tại đây, đạo hàm tại điểm này âm, nghĩa là giá trị đạo hàm đang hướng về bên trái, trong khi điểm cực tiểu lại nằm bên phải.

Hoặc nếu xét điểm **θ** bên phải, đạo hàm dương (hướng về phải), trong khi điểm cực tiểu lại nằm bên trái.

- Nhận định: dấu của đạo hàm "ngược hướng" điểm cực tiểu
- Do đó, để di chuyển về điểm cực tiểu ta phải cập nhật:

$$\theta < \theta \quad \frac{dL(\theta)}{d\theta}$$

- Vấn đề: Khi giá trị đạo hàm quá lớn thì khi cập nhật, θ sẽ đi quá đà.

Giải pháp: Nhân với hệ số learning rate α

$$\theta \leftarrow \theta - \alpha \cdot \frac{dL(\theta)}{d\theta}$$

Nếu **α** quá lớn: **θ** dao động, không hội tụ (nhảy qua lại điểm cực tiểu).

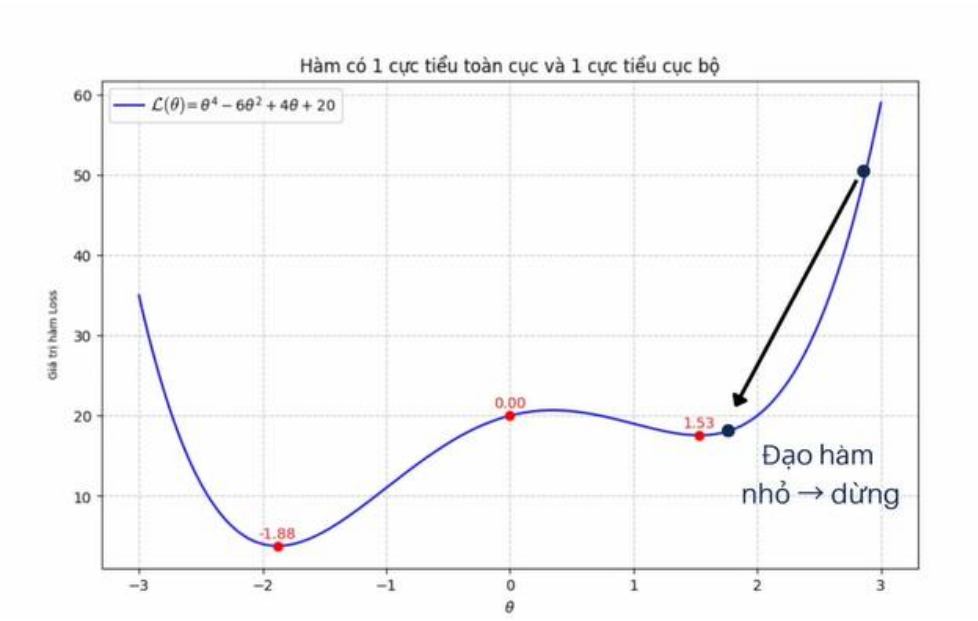
Nếu **α** quá nhỏ: **θ** đi từng bước rất nhỏ, tốn thời gian.

→ Cần điều chỉnh **α** thủ công hoặc dùng kỹ thuật tối ưu.

Khi nào dừng cập nhật θ?

1. Khi đạo hàm đủ nhỏ (đặt ngưỡng **ε**, ví dụ 10^{-4} hoặc 10^{-6}).
2. Dừng sau một số vòng lặp nhất định.

Mở rộng : Trường hợp hàm loss có nhiều điểm cực tiểu:



- **Cực tiểu cục bộ (local minimum)**
- **Cực tiểu toàn cục (global minimum)**:

Nếu khởi tạo **θ** gần cực tiểu cục bộ, thuật toán có thể mắc kẹt tại đó.

Giải pháp:

1. Chạy nhiều lần với **θ** khởi tạo khác nhau, chọn kết quả tốt nhất.
    a. *Ưu điểm*: Đơn giản.
    b. *Nhược điểm*: Tốn thời gian.
2. Dùng **Momentum**: Lợi dụng quán tính để vượt qua cực tiểu cục bộ.
3. Dùng THUẬT TOÁN **Adam** (kết hợp Momentum và điều chỉnh **α** tự động).

Cơ chế Momentum:

a. Không chỉ sử dụng gradient hiện tại mà còn tích lũy gradient từ các bước trước.
b. Giúp thuật toán vượt qua các điểm **local minimum** và **dao động** (oscillations) dễ dàng hơn.
c. Công thức cập nhật:

$v = \beta * v + (1 - \beta) * \nabla\theta$

$\theta = \theta - \eta * v$

Trong đó:

i. v: Vận tốc tích lũy (velocity)
ii. β: Hệ số momentum (0.9 trong code)
iii. $\nabla\theta$: Gradient hiện tại
iv. η: Tốc độ học (learning_rate = 0.02)

```
for i in range(N):
    # tính giá trị hàm loss
    loss = cal_loss_function_theta(theta)

    # tính giá trị đạo hàm của hàm loss theo theta
    grad = cal_loss_derivative_theta(theta)

    # Vẽ điểm theta hiện tại
    plt.plot(theta, loss, 'ro') # ro là điểm tròn màu đỏ

    # Dừng để quan sát
    plt.pause(0.1)

    # Tính vận tốc
    v = beta * v + (1 - beta) * grad

    # Cập nhật tham số theta dựa vào vận tốc
    theta = theta - learning_rate * v
```

- a. **Từng bước trong mỗi vòng lặp:**

2. **Tính loss**:
   a. Gọi hàm cal_loss_function_theta(theta) để tính giá trị hàm loss tại theta hiện tại.
   b. Mục đích: Theo dõi giá trị loss và hiển thị trên đồ thị.
3. **Tính gradient**:

   Gọi hàm cal_loss_derivative_theta(theta) để tính đạo hàM (gradient) tại theta.

   Gradient cho biết "hướng dốc nhất" của hàm loss. Ví dụ: Nếu grad > 0, cần giảm θ để giảm loss..

4. **Cập nhật vận tốc (quan trọng nhất)**:

   ```
   v = beta * v + (1 - beta) * grad
   ```

   a. beta * v: Giữ lại 90% (β=0.9) vận tốc từ bước trước → **duy trì quán tính**.
   b. (1 - beta) * grad: Thêm 10% ảnh hưởng của gradient hiện tại → **điều chỉnh hướng**.
   c. **Kết quả**: v là trung bình có trọng số của toàn bộ gradient trong quá khứ → giảm nhiễu và dao động.