



THUẬT TOÁN VÀ LÝ THUYẾT MÁY HỌC

HỒI QUY TUYẾN TÍNH VÀ GRADIENT DESCENT

TS. Lương Ngọc Hoàng



Nội dung

Giới thiệu Hồi quy tuyến tính đơn giản
Thuật toán Gradient Descent
Hệ số học
Gradient Descent và Xấp xỉ bậc hai



GIỚI THIỆU

INTRODUCTION



Ví dụ 1

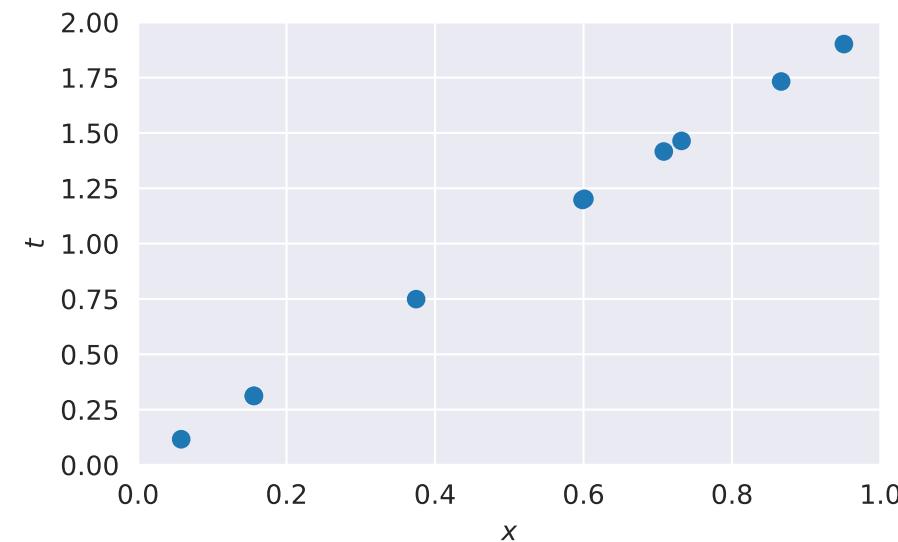
Tập dữ liệu (Dataset)

Phát sinh một tập dữ liệu (dataset) gồm N điểm dữ liệu (data points) $D = \{(x^{(i)}, t^{(i)})\}_{i=1}^N$ với:

- $x^{(i)} \in (0,1)$ là **đặc trưng đầu vào** (input feature) của điểm dữ liệu thứ i .
- $t^{(i)} \in \mathbb{R}$ là **giá trị đích** (target value) của điểm dữ liệu thứ i .

Giả sử $t^{(i)} \sim f(x^{(i)})$ với $f(x) = 2x$ là một **hàm đích không biết trước** (unknown target function).

Lưu ý: Trong ví dụ này, ta sử dụng hàm f để xây dựng tập dữ liệu minh họa. Trong thực tế, khi thu thập dữ liệu cho bài toán cần giải quyết, ta chỉ quan sát được giá trị đích $t^{(i)}$ của mỗi điểm dữ liệu với đặc trưng $x^{(i)}$ chứ không thể tính được $f(x^{(i)})$.



```
def f(x):  
    return 2 * x  
  
np.random.seed(42)  
x = np.random.uniform(0, 1, 10)  
t = f(x)
```

Ví dụ 1

Mô hình (Model)

Ta thực nghiệm với một mô hình đơn giản có dạng:

$$y = f_w(x) = w_0 x$$

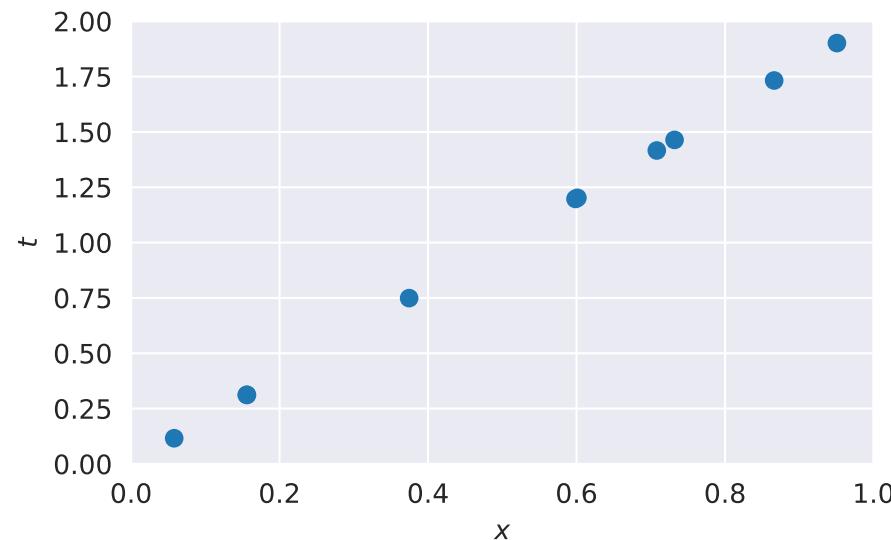
với w_0 là **tham số (parameter)** duy nhất của mô hình.

Lưu ý:

- *Hàm đích $f(x)$ là thông tin không có sẵn (unknown).*
- *Mô hình $f_w(x)$ là do ta xây dựng để xấp xỉ hàm đích $f(x)$ thông qua tập dữ liệu D .*

Ta muốn tìm giá trị của w_0 sao cho:

$$f_w(x^{(i)}) = w_0 x^{(i)} \approx t^{(i)}, \quad i = 1, \dots, N.$$



```
def model(x, w_0):  
    return w_0 * x
```



Ví dụ 1

Thuật toán (Algorithm)

$$y = f_w(x) = w_0x$$

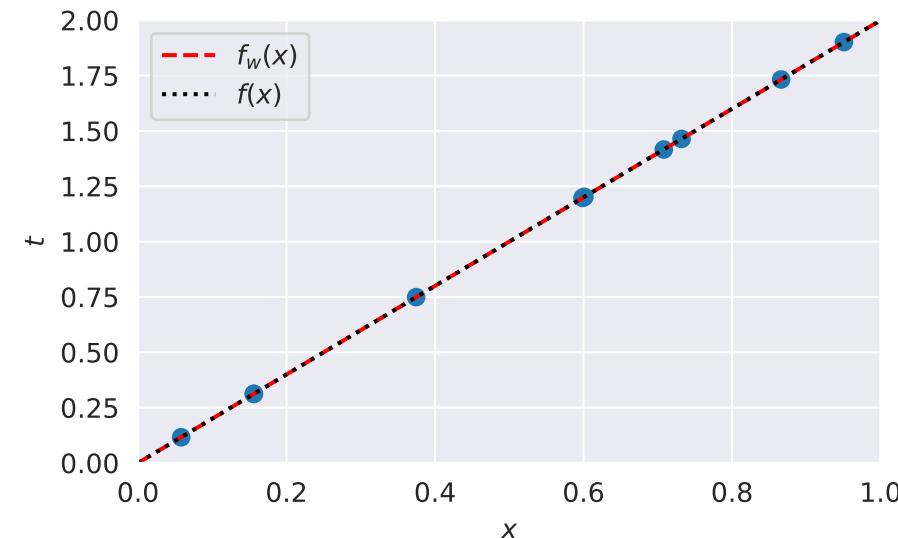
Trong ví dụ này, tất cả mọi điểm dữ liệu nằm trên một đường thẳng, ta có thể chọn bất kỳ 2 điểm dữ liệu $(x^{(i)}, t^{(i)})$ và $(x^{(k)}, t^{(k)})$ để tính độ dốc (slope) của đường thẳng.

$$w_0 = \frac{t^{(k)} - t^{(i)}}{x^{(k)} - x^{(i)}} = 2.0$$

Ta xây dựng được mô hình như sau:

$$f_w(x) = w_0x = 2x$$

Mô hình này dự đoán đúng giá trị đích $t^{(i)}$ của mọi điểm dữ liệu, và chính là hàm đích (target function) $f(x) = 2x$.



```
plt.plot([0, 1], [model(0, w_0), model(1, w_0)],  
'r--', label='$f_w(x)$')
```

```
plt.plot([0, 1], [f(0), f(1)], 'k:', label='$f(x)$')
```



Ví dụ 2

Tập dữ liệu (Dataset)

Phát sinh một tập dữ liệu (dataset) gồm N điểm dữ liệu (data points)

$$D = \{(x^{(i)}, t^{(i)})\}_{i=1}^N$$
 với:

- $x^{(i)} \in (0,1)$ là **đặc trưng đầu vào** (input feature) của điểm dữ liệu thứ i .
- $t^{(i)} \in \mathbb{R}$ là **giá trị đích** (target value) của điểm dữ liệu thứ i .

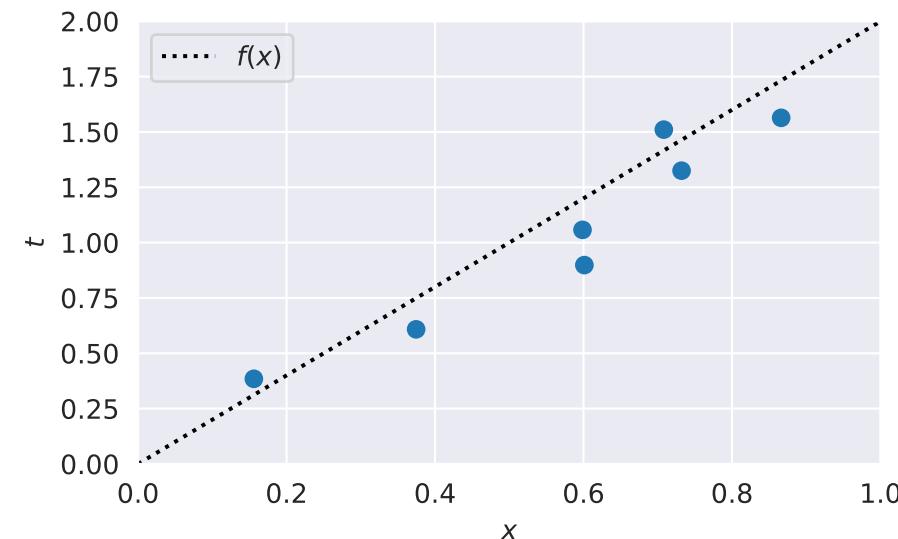
Trong thực tế, dữ liệu thu thập có thể có chứa **nhiễu (noise)** hoặc **sai số (error)** đến từ đo đạc hay quan sát. Trong ví dụ này, để mô phỏng những sai lệch đó, chúng ta cộng thêm một lượng nhiễu nhỏ vào các giá trị đích.

$$\varepsilon^{(i)} = \mathcal{N}(0, \sigma^2)$$

$$t^{(i)} = f(x^{(i)}) + \varepsilon^{(i)}$$

với $f(x) = 2x$ là một **hàm đích không biết trước** (unknown target function).

Lưu ý: Trong ví dụ này, ta sử dụng hàm f và tự phát sinh nhiễu ε để xây dựng tập dữ liệu minh họa. Trong thực tế, khi thu thập dữ liệu cho bài toán cần giải quyết, ta chỉ quan sát được giá trị đích $t^{(i)}$ của mỗi điểm dữ liệu với đặc trưng $x^{(i)}$ chứ không thể tính được $f(x^{(i)})$ và cũng không biết được điểm dữ liệu thu thập có chứa lượng nhiễu $\varepsilon^{(i)}$ tuân theo phân phối nào.



```
noise = np.random.normal(0, 0.3,  
size=x.shape[0])  
t = f(x) + noise
```



Ví dụ 2

Mô hình (Model)

Tương tự, ta thực nghiệm với một mô hình đơn giản có dạng:

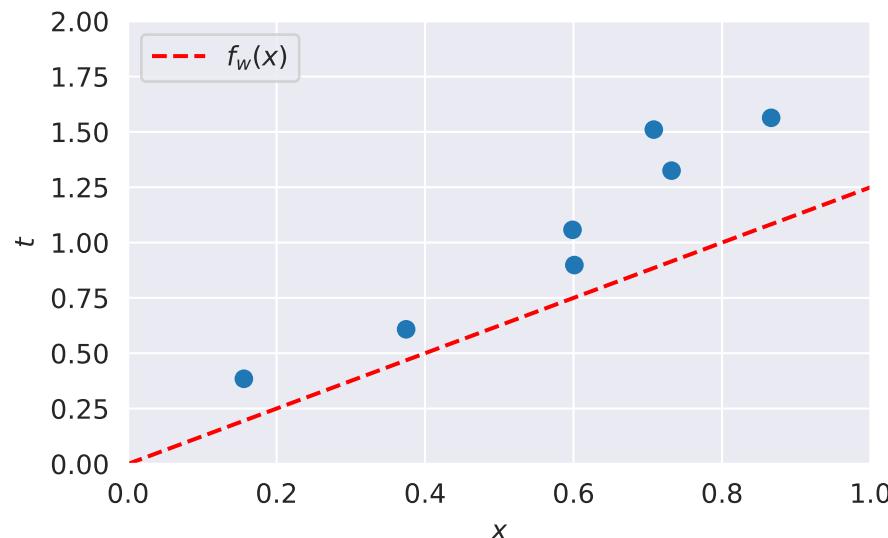
$$y = f_w(x) = w_0 x$$

với w_0 là **tham số (parameter)** duy nhất của mô hình.

Với tập dữ liệu mới này, ta không thể tính giá trị của w_0 như trong Ví dụ 1.

Ta có thể thử nhiều giá trị khác nhau cho w_0 như sau:

- $w_0 = 4.0$. Giá trị $w_0 = 4.0$ quá lớn.
- $w_0 = 3.0$. Giá trị $w_0 = 3.0$ vẫn tương đối lớn.
- $w_0 = 0.75$. Giá trị $w_0 = 0.75$ quá nhỏ.
- $w_0 = 1.25$. Giá trị $w_0 = 1.25$ vẫn tương đối nhỏ.
- ...



```
w_0 = 4.0 # 3.0, 0.75, 1.25, ...
plt.plot([0, 1], [model(0, w_0), model(1, w_0)],
'r--', label='$f_w(x)$')
```



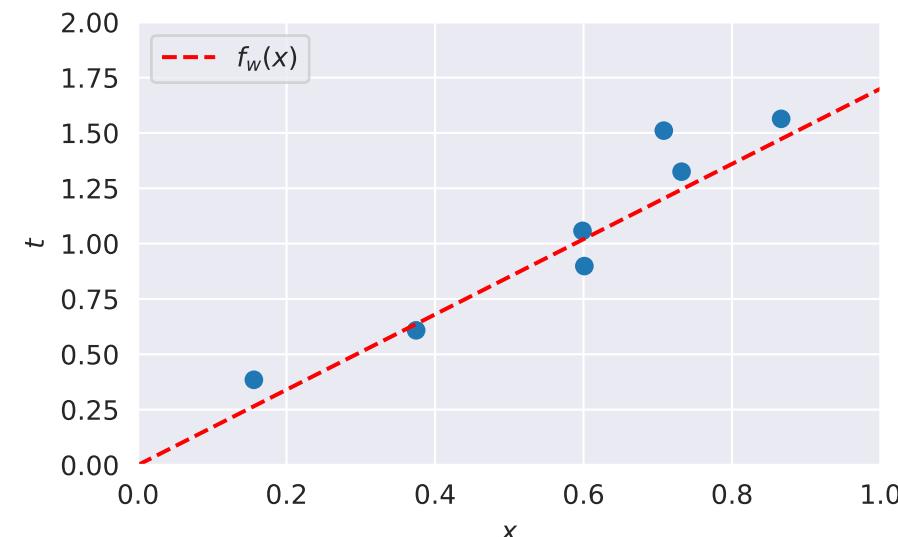
Ví dụ 2

• Hàm mất mát (Loss function)

Để đánh giá chất lượng của mô hình, ta định nghĩa một hàm mất mát (loss function) để đo sai số (error) dự đoán của mô hình $f_w(x^{(i)})$ với giá trị đích $t^{(i)}$ của mỗi điểm dữ liệu $(x^{(i)}, t^{(i)})$ trong tập dữ liệu D :

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N (t^{(i)} - f_w(x^{(i)}))^2$$

- Tuy nhiên, hàm mất mát trên không thật sự tốt.
- Mô hình có sai số âm trên một số điểm dữ liệu, và sai số dương trên một số điểm dữ liệu khác. Ta quan tâm đến **độ lớn của sai số (magnitude)** hơn là dấu của sai số (sign).



$$w_0 = 1.75$$

```
plt.plot([0, 1], [model(0, w_0), model(1, w_0)],  
'r--', label='f_w(x)')
```



Ví dụ 2

Hàm mất mát (Loss function)

- Để đánh giá chất lượng của mô hình, ta định nghĩa một hàm mất mát (loss function) để đo sai số dự đoán của mô hình $f_w(x^{(i)})$ với giá trị đích $t^{(i)}$ của mỗi điểm dữ liệu $(x^{(i)}, t^{(i)})$ trong tập dữ liệu D .

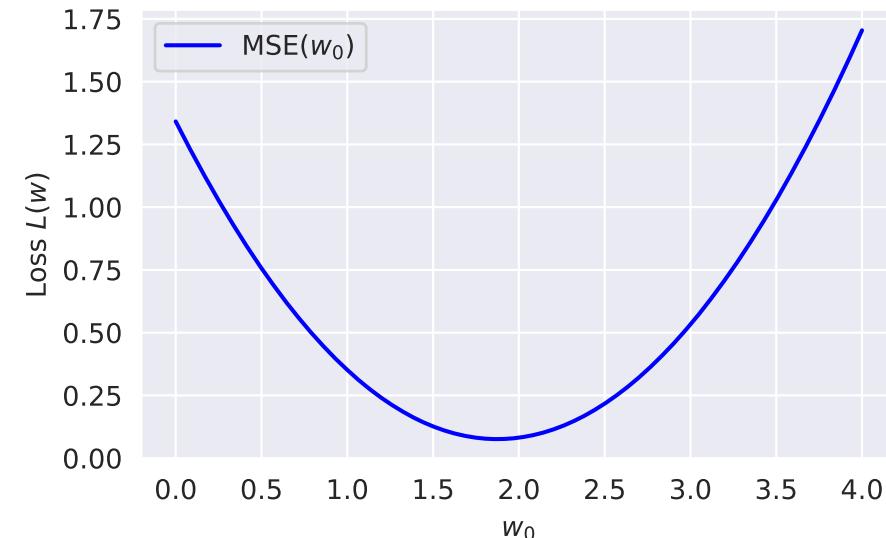
$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \left(t^{(i)} - f_{\mathbf{w}}(x^{(i)}) \right)^2$$

Hàm mất mát có tên là Mean Squared Error (MSE – Sai số bình phương trung bình).

Ta thử tính giá trị MSE cho nhiều giá trị khác nhau của w_0 :

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \left(t^{(i)} - w_0 x^{(i)} \right)^2$$

Ta thấy, do yếu tố nhiễu (noise) trong tập dữ liệu, giá trị w_0 tại đó hàm mất mát MSE đạt giá trị cực tiểu không phải là 0.



```
def loss(prediction, t):
    return np.mean((t - prediction)**2)
```

```
w_s = np.linspace(0, 4, num=1000) # w_0 values
losses = np.array([loss(model(x, w_0), t) for w_0 in w_s])
plt.plot(w_s, losses, 'b-', label=f'MSE({w_0})')
```



Ví dụ 2

Cực trị của hàm mất mát MSE:

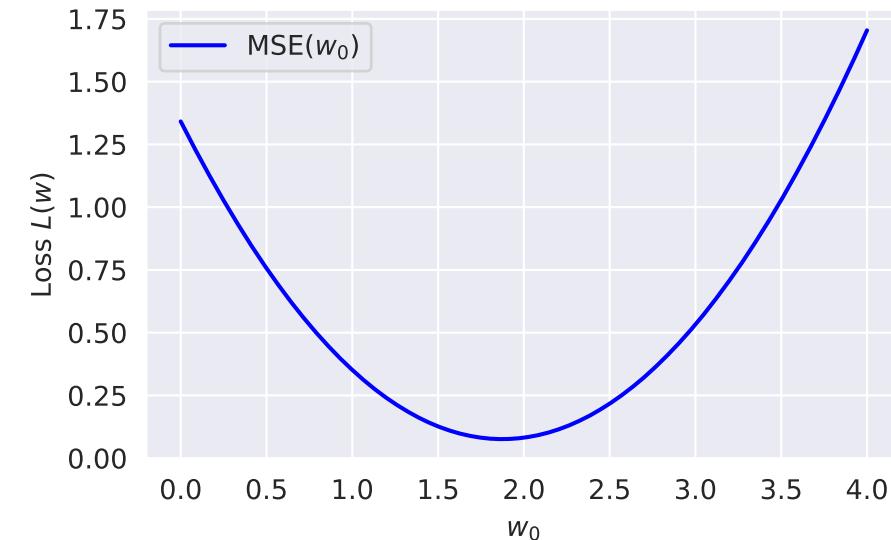
- Ta muốn xác định giá trị tối ưu của w_0 mà tại đó hàm mất mát $L(\mathbf{w}) = \text{MSE}(\mathbf{w}_0)$ đạt giá trị nhỏ nhất.
- Đây là giá trị để mô hình $f_{\mathbf{w}}(x) = w_0 x$ đạt được độ lỗi bình phương trung bình nhỏ nhất xét trên các điểm dữ liệu trong tập dữ liệu D :

$$w_0^* = \arg \min_{w_0} L(\mathbf{w}) = \arg \min_{w_0} \frac{1}{N} \sum_{i=1}^N (t^{(i)} - w_0 x^{(i)})^2$$

Ta có thể tìm w_0^* bằng cách giải $\frac{dL}{dw_0} = 0$ với:

$$\frac{dL}{dw_0} = \frac{d}{dw_0} \left(\frac{1}{N} \sum_{i=1}^N (t^{(i)} - w_0 x^{(i)})^2 \right) = \frac{1}{N} \sum_{i=1}^N \frac{d}{dw_0} (t^{(i)} - w_0 x^{(i)})^2$$

$$= \frac{1}{N} \sum_{i=1}^N 2(t^{(i)} - w_0 x^{(i)}) (-x^{(i)}) = \frac{1}{N} \sum_{i=1}^N 2(w_0 x^{(i)} - t^{(i)}) x^{(i)}$$



```
def loss(prediction, t):
    return np.mean((t - prediction)**2)
```

```
w_s = np.linspace(0, 4, num=1000) # w_0 values
losses = np.array([loss(model(x, w_0), t) for w_0 in w_s])
plt.plot(w_s, losses, 'b-', label=f'MSE({w_0})')
```



THUẬT TOÁN GRADIENT DESCENT

GRADIENT DESCENT ALGORITHM



Thuật toán Gradient Descent

Đầu vào: Hàm số $f(x)$ cần cực tiểu hóa.

Giá trị khởi tạo x_0 tại $t = 0$ (initial value).

Hệ số học (learning rate) $\alpha > 0$.

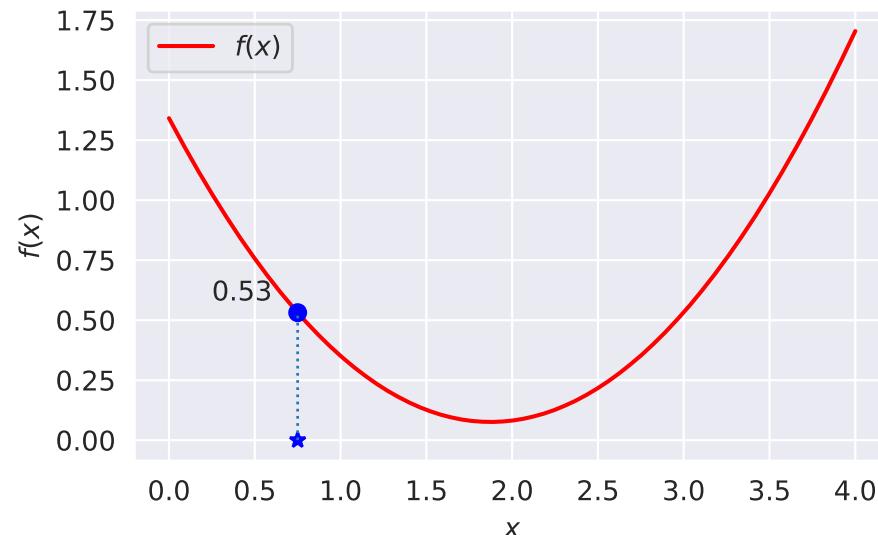
while [*điều kiện dừng chưa thỏa*] **do**:

$$g_t = f'(x_t)$$

$$x_{t+1} = x_t - \alpha g_t$$

$$t = t + 1$$

- Ở mỗi bước t , ta tính đạo hàm $f'(x_t)$ của hàm số f tại giá trị x_t .
- Ta cập nhật giá trị x theo chiều **ngược** với đạo hàm và được điều chỉnh với hệ số học α .
- Nếu đạo hàm khác 0, hàm số f thỏa mãn một số điều kiện về độ trơn (smoothness), và hệ số học α đủ nhỏ thì ta sẽ có x_{t+1} tốt hơn x_t trong việc cực tiểu hóa hàm số $f(x)$, tức là $f(x_{t+1}) < f(x_t)$.
- Thuật toán dừng lại khi một điều kiện dừng nào đó được thỏa:
 - ✓ Số bước t của thuật toán đạt ngưỡng tối đa cho phép t_{\max} .
 - ✓ Độ lớn của đạo hàm nhỏ hơn một ngưỡng $\varepsilon > 0$ cho trước $|g_t| < \varepsilon$.
 - ✓ ...



Gradient Descent – Cực tiểu hóa hàm MSE



Đầu vào: Hàm mất mát $L(\mathbf{w}) = \text{MSE}(\mathbf{w}_0)$ cần cực tiểu hóa.

Giá trị khởi tạo \mathbf{w}_0 tại $t = 0$ (initial value).

Hệ số học (learning rate) $\alpha > 0$.

while [*điều kiện dừng chưa thỏa*] **do**:

$$g_t = L'(\mathbf{w}_0)$$

$$\mathbf{w}_0 = \mathbf{w}_0 - \alpha g_t$$

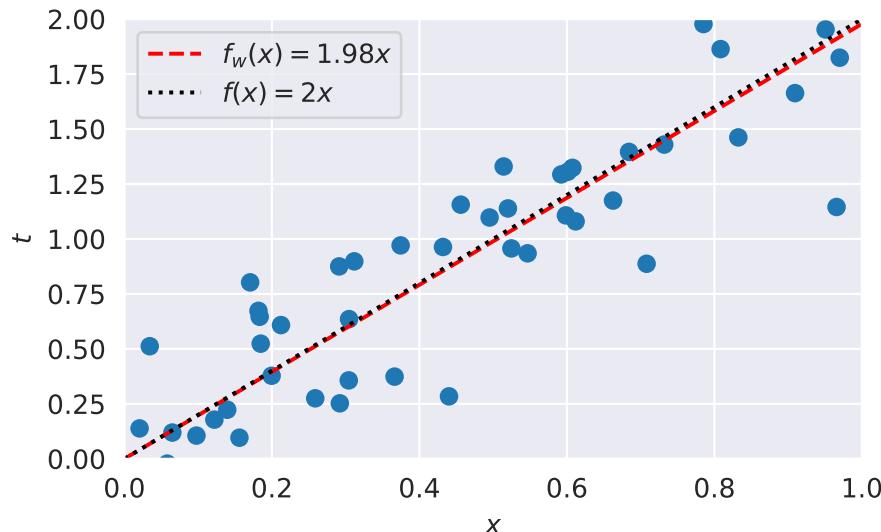
$$t = t + 1$$

Ta muốn tìm giá trị của \mathbf{w}_0 có thể cực tiểu hóa hàm mất mát MSE:

$$\mathbf{w}_0^* = \arg \min_{\mathbf{w}_0} L(\mathbf{w}) = \arg \min_{\mathbf{w}_0} \frac{1}{N} \sum_{i=1}^N (t^{(i)} - \mathbf{w}_0 \cdot \mathbf{x}^{(i)})^2$$

Đạo hàm của hàm mất mát MSE theo \mathbf{w}_0 là:

$$\frac{dL}{d\mathbf{w}_0} = \frac{1}{N} \sum_{i=1}^N 2(\mathbf{w}_0 \cdot \mathbf{x}^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$



```
def derivative(w_0, x, t):  
    return np.mean(2 * x * (model(x, w_0) - t))
```

```
w_0 = np.random.rand()  
alpha = 0.9  
nb_of_iterations = 4
```

```
for i in range(nb_of_iterations):  
    g = derivative(w_0, x, t)  
    w_0 = w_0 - alpha * g
```



Ví dụ 3

Tập dữ liệu (Dataset)

- Phát sinh một tập dữ liệu (dataset) gồm N điểm dữ liệu (data points)

$D = \{(x^{(i)}, t^{(i)})\}_{i=1}^N$ với:

- $x^{(i)} \in (0,1)$ là **đặc trưng đầu vào** (input feature) của điểm dữ liệu thứ i .
- $t^{(i)} \in \mathbb{R}$ là **giá trị đích** (target value) của điểm dữ liệu thứ i .

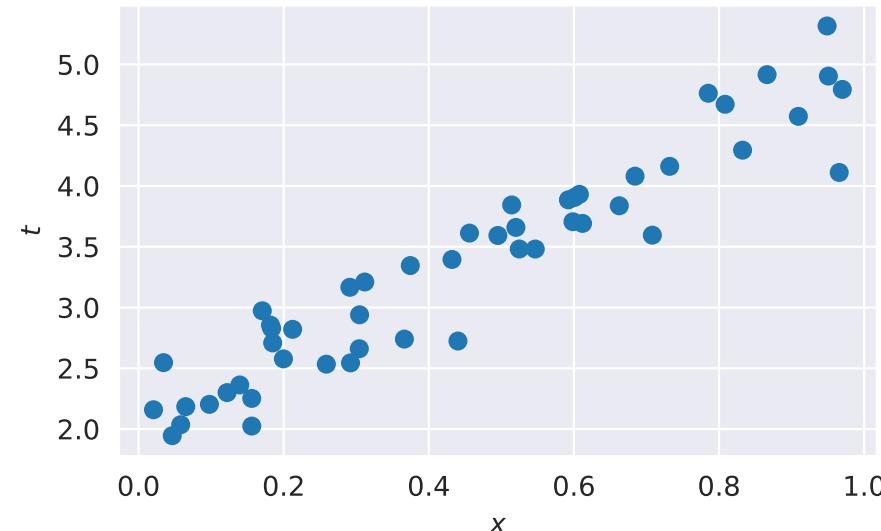
Trong ví dụ này, các giá trị đích có chứa một lượng nhiễu (noise) nhỏ:

$$\varepsilon^{(i)} = \mathcal{N}(0, \sigma^2)$$

$$t^{(i)} = f(x^{(i)}) + \varepsilon^{(i)}$$

với $f(x) = 3x + 2$ là một **hàm đích không biết trước** (unknown target function).

Lưu ý: Trong ví dụ này, ta sử dụng hàm f và tự phát sinh nhiễu ε để xây dựng tập dữ liệu minh họa. Trong thực tế, khi thu thập dữ liệu cho bài toán cần giải quyết, ta chỉ quan sát được giá trị đích $t^{(i)}$ của mỗi điểm dữ liệu với đặc trưng $x^{(i)}$ chứ không thể tính được $f(x^{(i)})$ và cũng không biết được điểm dữ liệu thu thập có chứa lượng nhiễu $\varepsilon^{(i)}$ tuân theo phân phối nào.



```
def f(x):
    return 3 * x + 2

np.random.seed(42)
x = np.random.uniform(0, 1, 50)

noise = np.random.normal(0, 0.3,
size=x.shape[0])
t = f(x) + noise
```



Ví dụ 3

Mô hình (Model)

- Tương tự, ta thực nghiệm với một mô hình đơn giản có dạng:

$$y = f_w(x) = w_0 x$$

với w_0 là tham số (parameter) duy nhất của mô hình.

Hàm mất mát (Loss function)

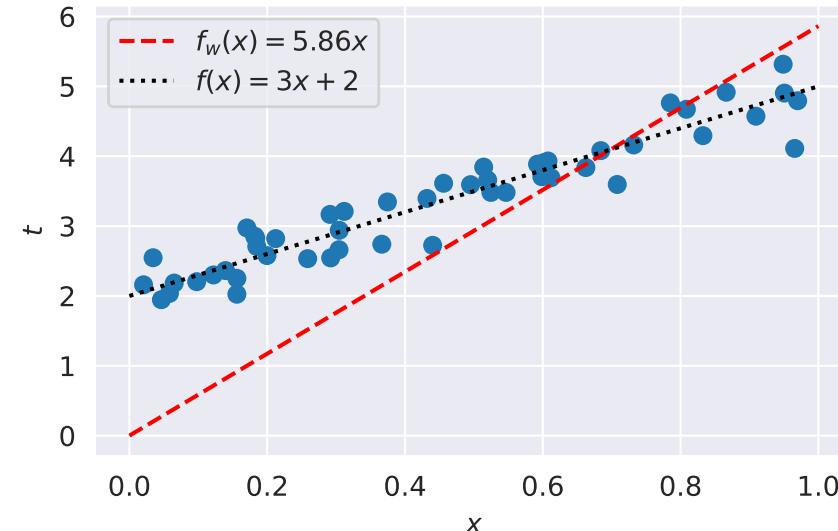
Ta sử dụng hàm mất mát MSE để đánh giá chất lượng của mô hình:

$$L(w) = \frac{1}{N} \sum_{i=1}^N (t^{(i)} - w_0 x^{(i)})^2$$

Tối ưu hóa (Optimization)

Ta sử dụng thuật toán Gradient Descent để tìm giá trị của tham số w_0 sao cho có thể cực tiểu hóa hàm mất mát MSE. Ở mỗi vòng lặp thứ t , ta cập nhật:

$$w_0^{(t+1)} = w_0^{(t)} - \alpha L'(w_0^{(t)})$$



```
def derivative(w_0, x, t):
    return np.mean(2 * x * (model(x, w_0) - t))

w_0 = np.random.rand()
alpha = 0.1
nb_of_iterations = 50
for i in range(nb_of_iterations):
    g = derivative(w_0, x, t)
    w_0 = w_0 - alpha * grad
```



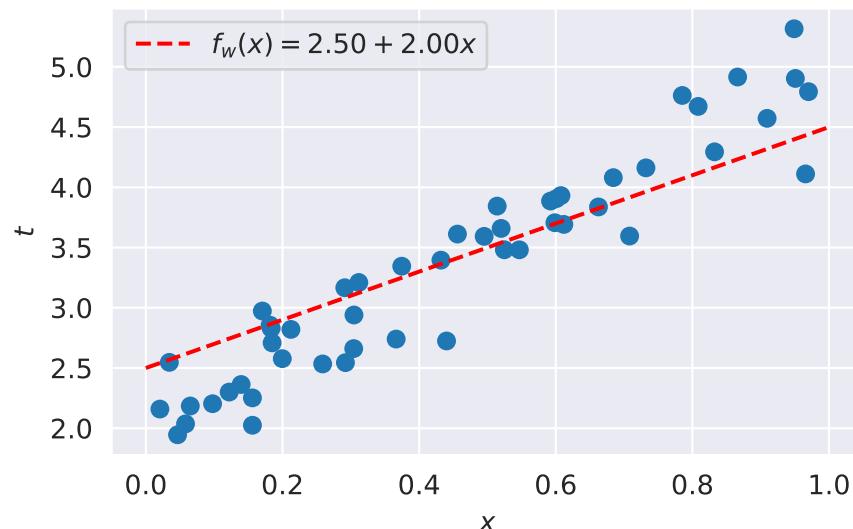
Ví dụ 4

Mô hình (Model)

- Ta thực nghiệm với một mô hình mới:

$$y = f_w(x) = w_0 + w_1 x$$

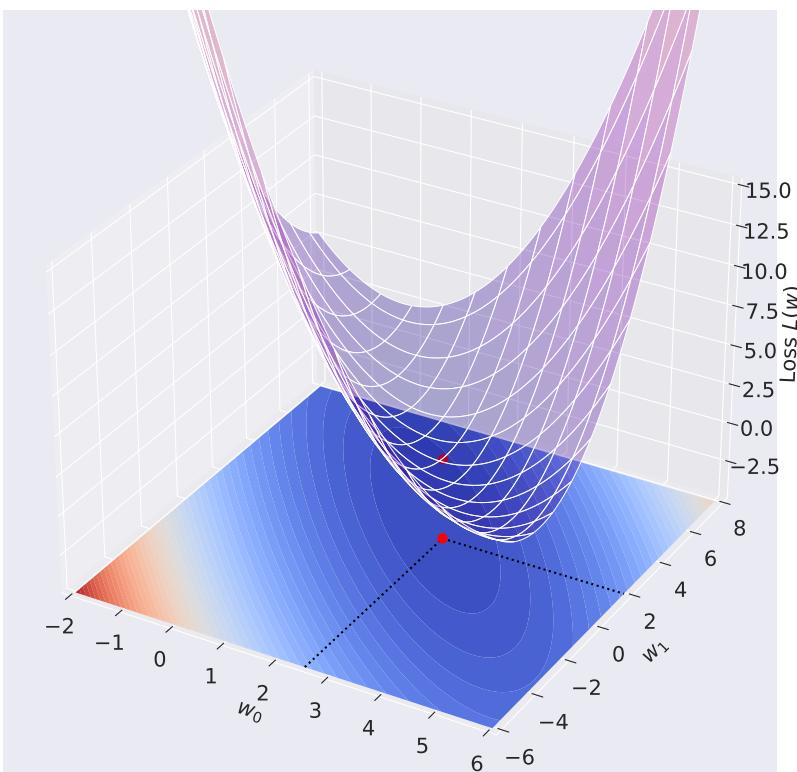
với $w = (w_0, w_1)$ là các tham số (parameters) của mô hình.



Hàm mất mát (Loss function)

Ta sử dụng hàm mất mát MSE để đánh giá chất lượng của mô hình:

$$\begin{aligned} L(w) &= \frac{1}{N} \sum_{i=1}^N \left(t^{(i)} - f_w(x^{(i)}) \right)^2 \\ &= \frac{1}{N} \sum_{i=1}^N \left(t^{(i)} - (w_0 + w_1 x^{(i)}) \right)^2 \end{aligned}$$





Ví dụ 4

Tối ưu hóa (Optimization)

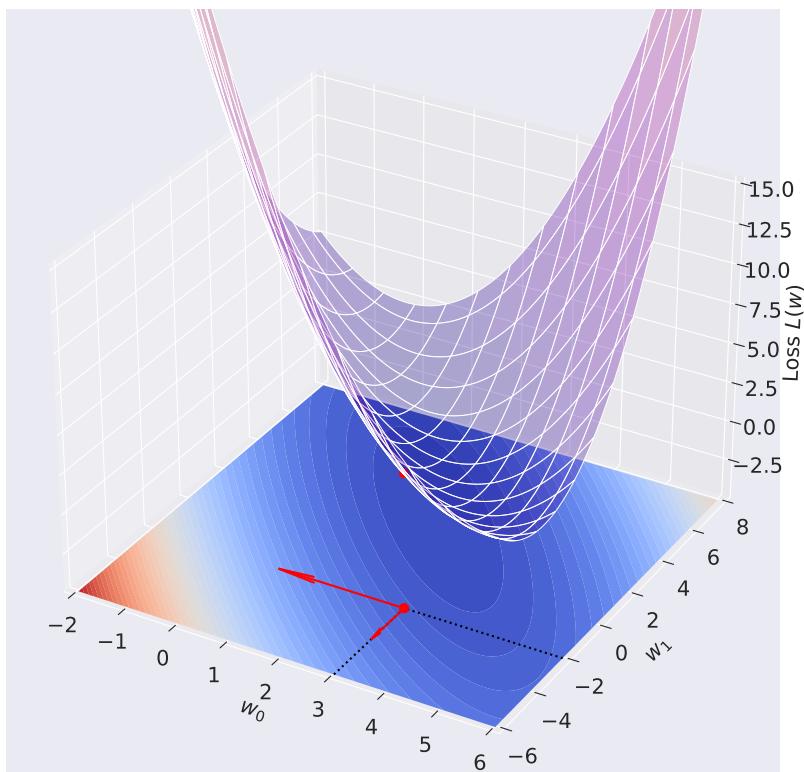
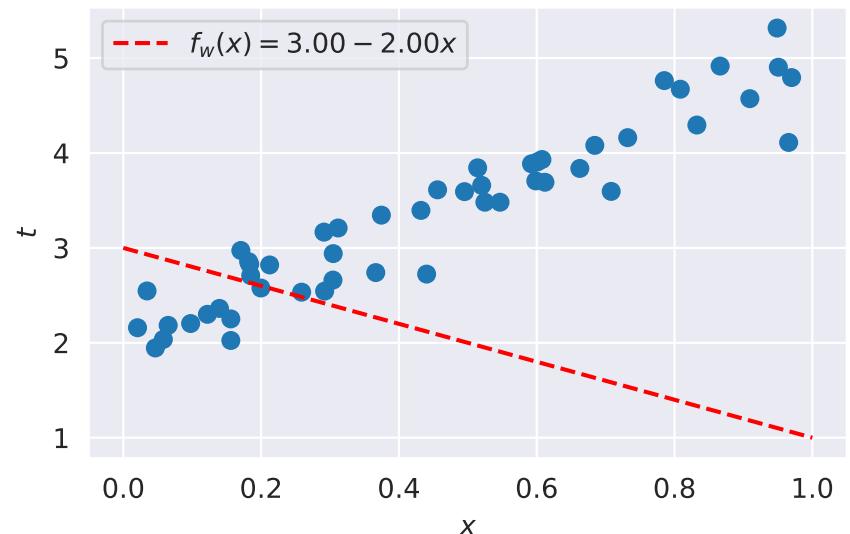
- Ta sử dụng thuật toán Gradient Descent để tìm giá trị của tham số w_0, w_1 sao cho có thể cực tiểu hóa hàm mất mát MSE.

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \left(t^{(i)} - f_{\mathbf{w}}(x^{(i)}) \right)^2 = \frac{1}{N} \sum_{i=1}^N \left(t^{(i)} - (w_0 + w_1 x^{(i)}) \right)^2$$

Ta tính các đạo hàm riêng (partial derivatives) của hàm mất mát theo w_0 và w_1 :

$$\frac{\partial L(\mathbf{w})}{\partial w_0} = \frac{1}{N} \sum_{i=1}^N \frac{\partial}{\partial w_0} \left(t^{(i)} - (w_0 + w_1 x^{(i)}) \right)^2 = \frac{1}{N} \sum_{i=1}^N 2 \left((w_0 + w_1 x^{(i)}) - t^{(i)} \right)$$

$$\frac{\partial L(\mathbf{w})}{\partial w_1} = \frac{1}{N} \sum_{i=1}^N \frac{\partial}{\partial w_1} \left(t^{(i)} - (w_0 + w_1 x^{(i)}) \right)^2 = \frac{1}{N} \sum_{i=1}^N 2 \left((w_0 + w_1 x^{(i)}) - t^{(i)} \right) x^{(i)}$$





Ví dụ 4

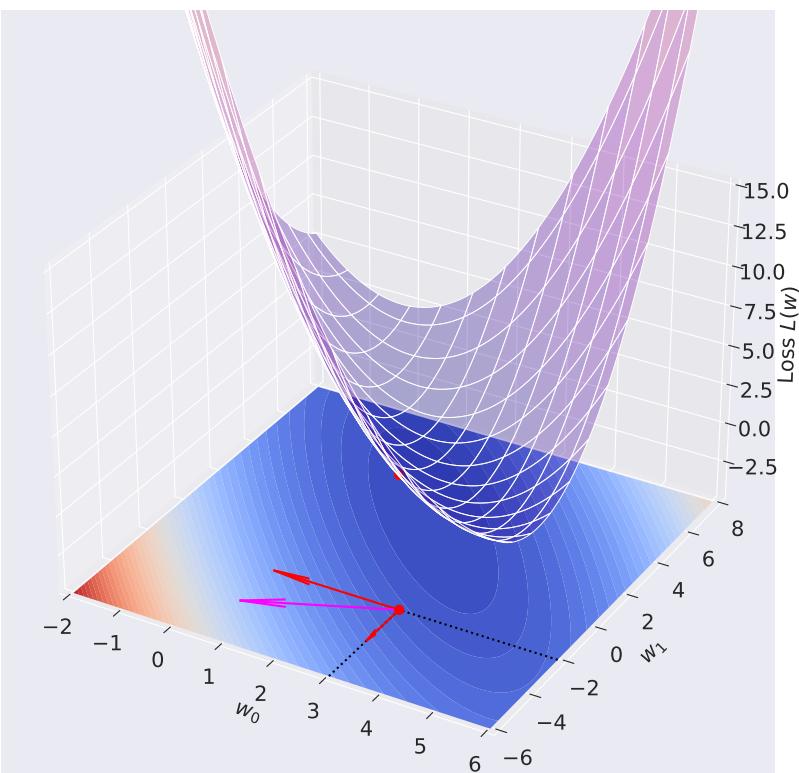
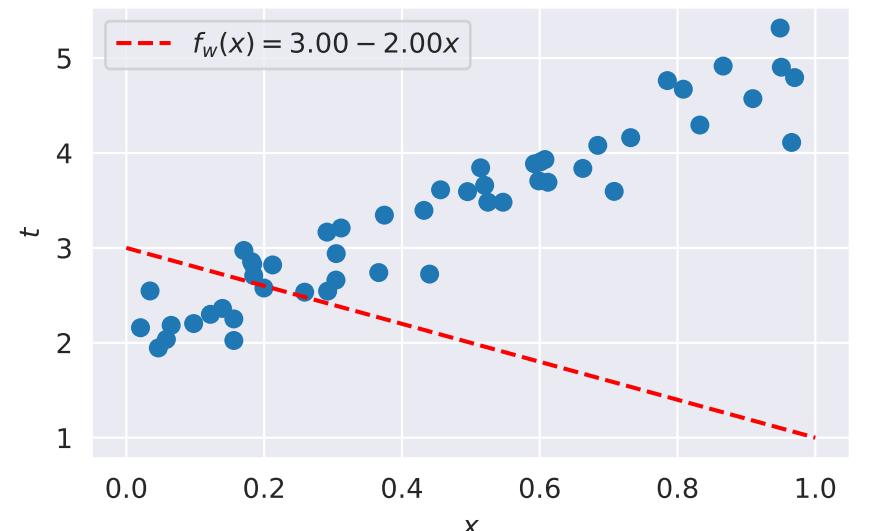
Gradient

- Định nghĩa **gradient** của hàm mất mát $L(\mathbf{w})$ là vector với thành phần là các đạo hàm riêng của hàm mất mát $L(\mathbf{w})$ theo từng tham số w_0, w_1 :

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \begin{bmatrix} \frac{\partial L(\mathbf{w})}{\partial w_0} \\ \frac{\partial L(\mathbf{w})}{\partial w_1} \end{bmatrix} = \frac{2}{N} \begin{bmatrix} \sum_{i=1}^N ((w_0 + w_1 x^{(i)}) - t^{(i)}) \cdot \mathbf{1} \\ \sum_{i=1}^N ((w_0 + w_1 x^{(i)}) - t^{(i)}) \cdot \mathbf{x}^{(i)} \end{bmatrix} = \frac{2}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{t})$$

với:

$$\mathbf{X} = \begin{bmatrix} 1 & \mathbf{x}^{(1)} \\ 1 & \mathbf{x}^{(2)} \\ \dots & \dots \\ 1 & \mathbf{x}^{(N)} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t^{(1)} \\ t^{(2)} \\ \dots \\ t^{(N)} \end{bmatrix}$$





Ví dụ 4

$$\nabla_w L(\mathbf{w}) = \begin{bmatrix} \frac{\partial L(\mathbf{w})}{\partial w_0} \\ \frac{\partial L(\mathbf{w})}{\partial w_1} \end{bmatrix} = \frac{2}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{t})$$

Gradient $\nabla_w L(\mathbf{w})$ là vector trỏ về hướng giá trị hàm mất mát $L(\mathbf{w})$ tăng. Ta đang cần cực tiểu hóa hàm mất mát, nên ta sẽ cập nhật giá trị các tham số $\mathbf{w} = (w_0, w_1)$ theo **hướng ngược lại** của gradient: $-\nabla_w L(\mathbf{w})$.

Đầu vào: Hàm mất mát $L(\mathbf{w}) = \text{MSE}(\mathbf{w}_0)$ cần cực tiểu hóa.

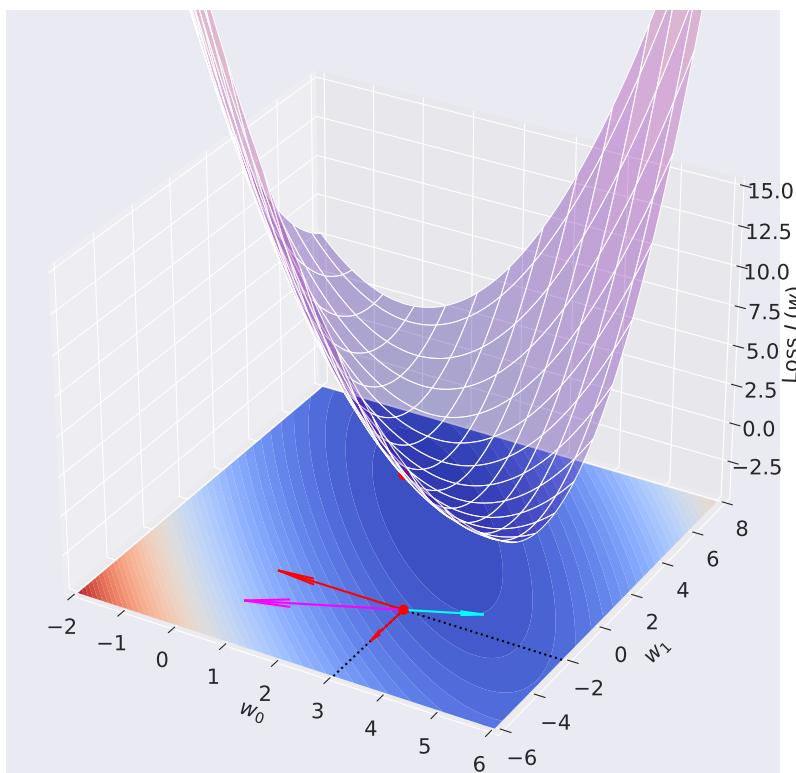
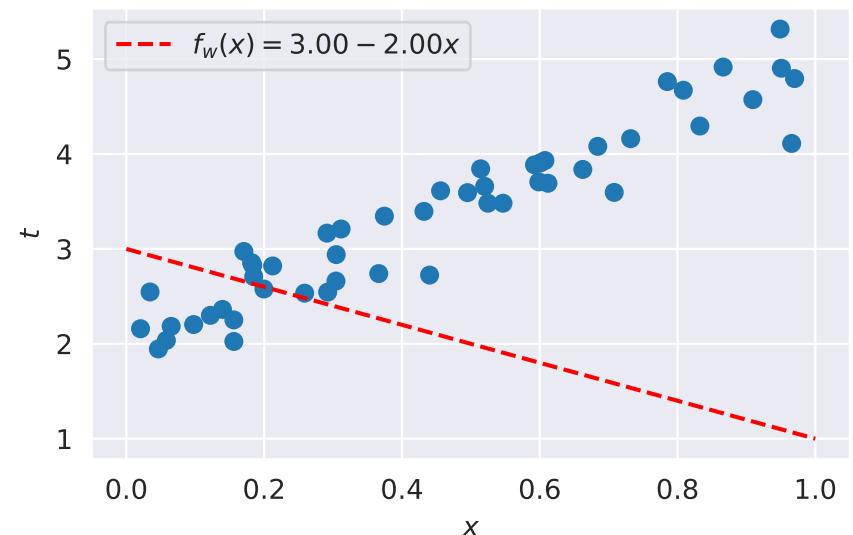
Giá trị khởi tạo \mathbf{w} tại $t = 0$ (initial value).

Hệ số học (learning rate) $\alpha > 0$.

while [*điều kiện dừng chưa thỏa*] **do**:

$$\mathbf{g}_t = \nabla_w L(\mathbf{w})$$

$$\mathbf{w} = \mathbf{w} - \alpha \mathbf{g}_t$$





Thuật toán Gradient Descent

Đầu vào: Hàm mất mát $L(\mathbf{w})$ cần cực tiểu hóa.

Giá trị khởi tạo \mathbf{w} tại $t = 0$ (initial value).

Hệ số học (learning rate) $\alpha > 0$.

while [*điều kiện dừng chưa thỏa*] **do**:

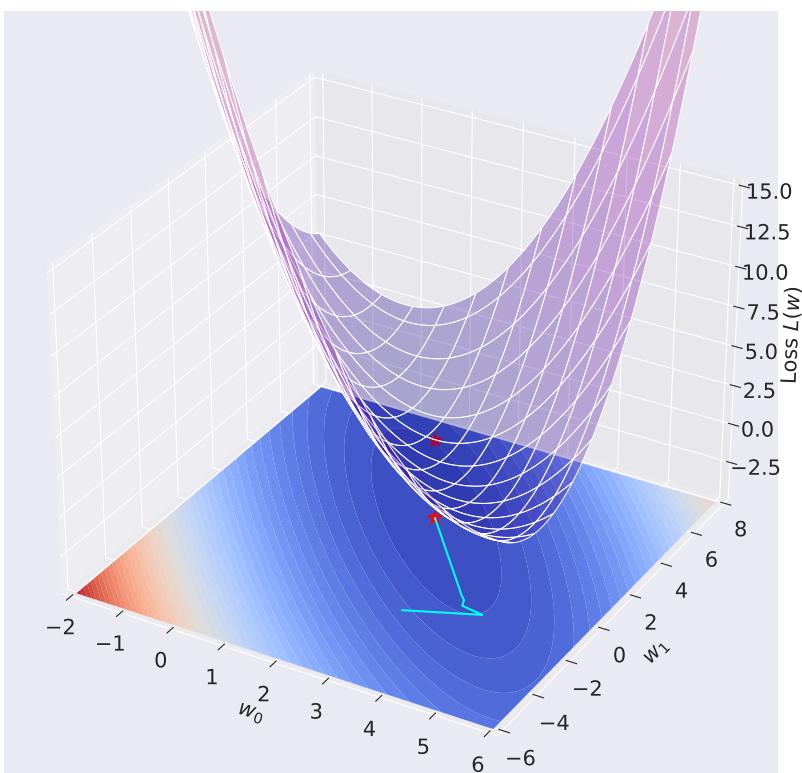
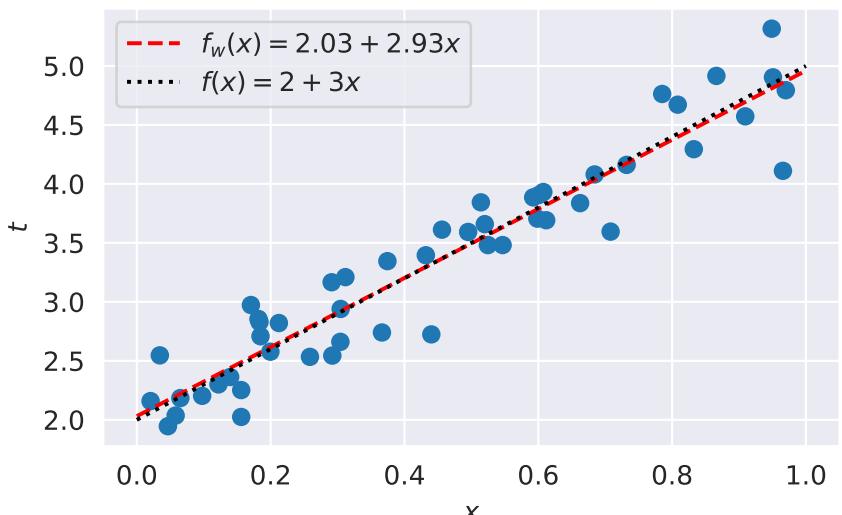
$$\mathbf{g}_t = \nabla_{\mathbf{w}} L(\mathbf{w})$$

$$\mathbf{w} = \mathbf{w} - \alpha \mathbf{g}_t$$

$$t = t + 1$$

```
def model(X, w):
    return X @ w
def gradient(w, X, t):
    return (2.0/X.shape[0]) * X.T @ (model(X, w) - t)

w = np.array([3,-2])
alpha = 0.5
nb_iterations = 100
for i in range(nb_iterations):
    grad = gradient(w, X, t)
    w = w - alpha * grad
```





HỆ SỐ HỌC

LEARNING RATE

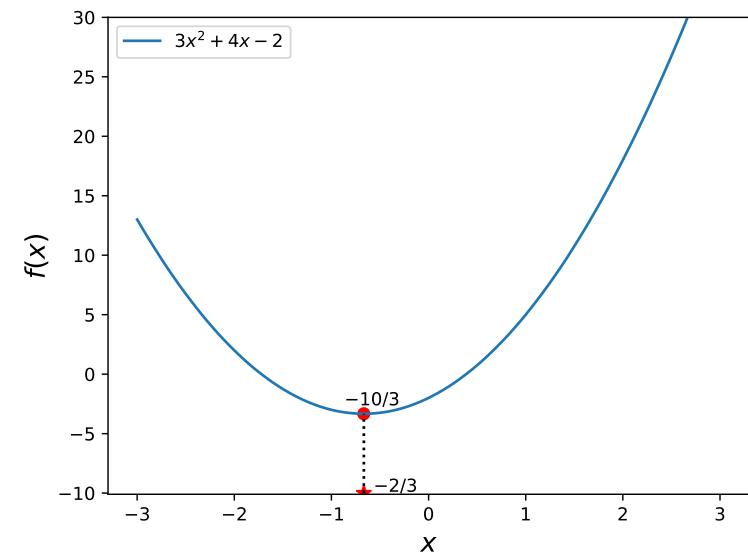


Ví dụ 5

- Cực tiểu hóa hàm số $f(x) = 3x^2 + 4x - 2$.
- Ta có thể dễ dàng giải phương trình đạo hàm $f'(x) = 6x + 4 = 0$, và suy ra lời giải $x^* = -2/3$.
- Sử dụng thuật toán Gradient Descent:
 - Chọn hệ số học (learning rate) $\alpha > 0$.
 - Khởi tạo x_t ngẫu nhiên tại bước $t = 1$.
 - Ở mỗi bước thứ t , ta thực hiện phép cập nhật:

$$x_{t+1} = x_t - \alpha f'(x_t)$$

- Thuật toán có thể hội tụ về lời giải tối ưu $x^* = -2/3$?





Ví dụ 5

- Ở mỗi bước thứ t , ta thực hiện phép cập nhật:

$$\begin{aligned}x_{t+1} &= x_t - \alpha f'(x_t) \\&= x_t - \alpha(6x_t + 4)\end{aligned}$$

Đặt $r = 1 - 6\alpha$. Ta có:

$$S = r^{t-1} + r^{t-2} + \cdots + r^1 + 1$$

$$rS = r^t + r^{t-1} + \cdots + r^2 + r^1$$

$$S - rS = 1 - r^t$$

$$S = \frac{1 - r^t}{1 - r} = \frac{1 - (1 - 6\alpha)^t}{1 - (1 - 6\alpha)}$$



Ví dụ 5

- Ở mỗi bước thứ t , ta thực hiện phép cập nhật:

$$\begin{aligned}x_{t+1} &= x_t - \alpha f'(x_t) \\&= x_t - \alpha(6x_t + 4) \\&= (1 - 6\alpha)^t x_1 - [(1 - 6\alpha)^{t-1} + (1 - 6\alpha)^{t-2} + \dots + (1 - 6\alpha) + 1]4\alpha\end{aligned}$$

Nếu $|1 - 6\alpha| < 1$ thì khi $t \rightarrow \infty$, ta có $(1 - 6\alpha)^t \rightarrow 0$. Như vậy ta có $x_t \rightarrow -\frac{2}{3}$.

Vậy điều kiện để thuật toán hội tụ là **hệ số học (learning rate)** $\alpha < \frac{1}{3}$.

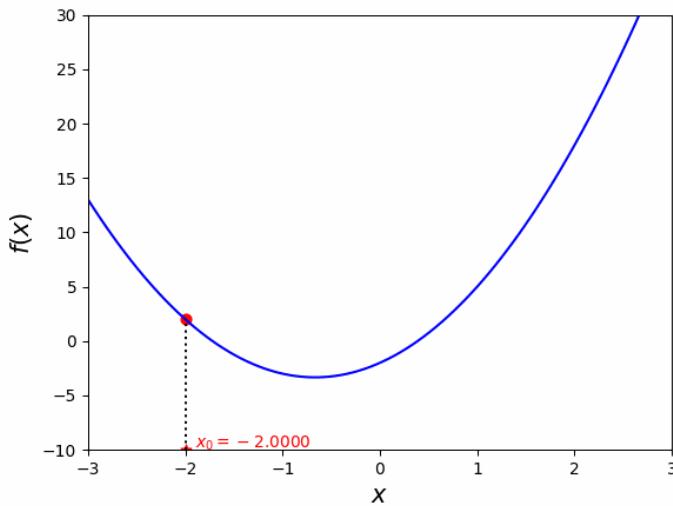


Hệ số học (learning rate)

- Sử dụng Gradient Descent cực tiểu hóa hàm số:

$$f(x) = 3x^2 + 4x - 2$$

- Ta thực nghiệm với giá trị khởi tạo $x = -2$.
 - Khi hệ số học $\alpha = 1/20$.
 - Khi hệ số học $\alpha = 1/10$.
 - Khi hệ số học $\alpha = 1/5$.
 - Khi hệ số học $\alpha = 1/3$.
 - Thuật toán không thể hội tụ do hệ số học quá lớn.
 - Khi hệ số học $\alpha = 1/6$.
 - Thuật toán hội tụ chỉ trong 1 bước.
 - Thông thường rất khó để xác định hệ số học tối ưu trong trường hợp tổng quát. Do đó, ta thường chọn hệ số học là một giá trị nhỏ để thuật toán có thể hội tụ.



```
def gradient(x):  
    return 6.0 * x + 4.0  
  
def gradient_descent(x_init, alpha, max_iters=100):  
    x = x_init  
    while True:  
        grad = gradient(x)  
        x = x - alpha * grad  
        if iter > max_iters:  
            break  
        if np.fabs(grad) < 1e-5:  
            break
```



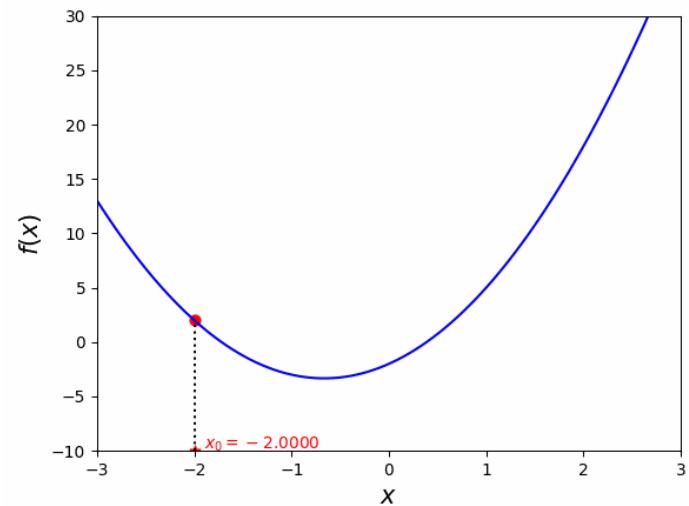
GRADIENT DESCENT VÀ XẤP XỈ BẬC HAI

GRADIENT DESCENT AND QUADRATIC APPROXIMATION



Thuật toán Gradient Descent

- Xét một hàm số $f(x)$ là **hàm lồi (convex)** và **khả vi (differentiable)** với miền xác định $dom(f) = \mathbb{R}^n$. Gọi x^* là lời giải cực tiểu của $f(x)$.
- Thuật toán **Gradient Descent** giải bài toán tối ưu như sau:
 1. Chọn điểm bắt đầu $x_0 \in \mathbb{R}^n$.
 2. $x_t = x_{t-1} - \alpha \nabla f(x_{t-1})$
 3. Lặp lại với $t = 1, 2, 3, \dots$
 4. Dừng lại khi thỏa điều kiện dừng nào đó.
 - Thuật toán thực hiện các bước lặp *xuống dốc (downhill)*.
 - Hướng ngược lại của gradient là hướng giảm của hàm số f .
 - Thuật toán dừng lại tại một điểm nào đó gần với điểm cực tiểu với mọi điểm bắt đầu bất kỳ. Lưu ý: chỉ đúng với các hàm lồi (convex).
 - Đối với các hàm không lồi (non-convex), tùy vào điểm bắt đầu x_0 mà ta có thể đạt được các lời giải cực tiểu địa phương (local minima) khác nhau.





Gradient Descent và Xấp xỉ bậc hai

- Ta có thể phân tích thuật toán Gradient Descent dưới góc độ một phương pháp **xấp xỉ bậc hai** (quadratic approximation) cho hàm số f như sau:
- Tại một điểm x bất kỳ, ta thực hiện **Khai triển Taylor bậc 1** của hàm số $f(y)$:

$$f(y) \approx f(x) + \nabla f(x) \cdot (y - x)$$

- Xấp xỉ trên chỉ tốt với những vị trí y gần với x . Ta giới hạn khoảng cách tới x như sau (với $\alpha > 0$):

$$f(y) \approx f(x) + \nabla f(x) \cdot (y - x) + \frac{1}{2\alpha} \|y - x\|_2^2$$

- Ta có một **xấp xỉ bậc hai** (quadratic approximation) của hàm số f .



Gradient Descent và Xấp xỉ bậc hai

- $f(\mathbf{y}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) + \frac{1}{2\alpha} \|\mathbf{y} - \mathbf{x}\|_2^2$
- Thuật toán Gradient Descent sẽ chọn điểm kế tiếp $\mathbf{y} = \mathbf{x}^+$ sao cho cực tiểu hóa được xấp xỉ bậc hai này:

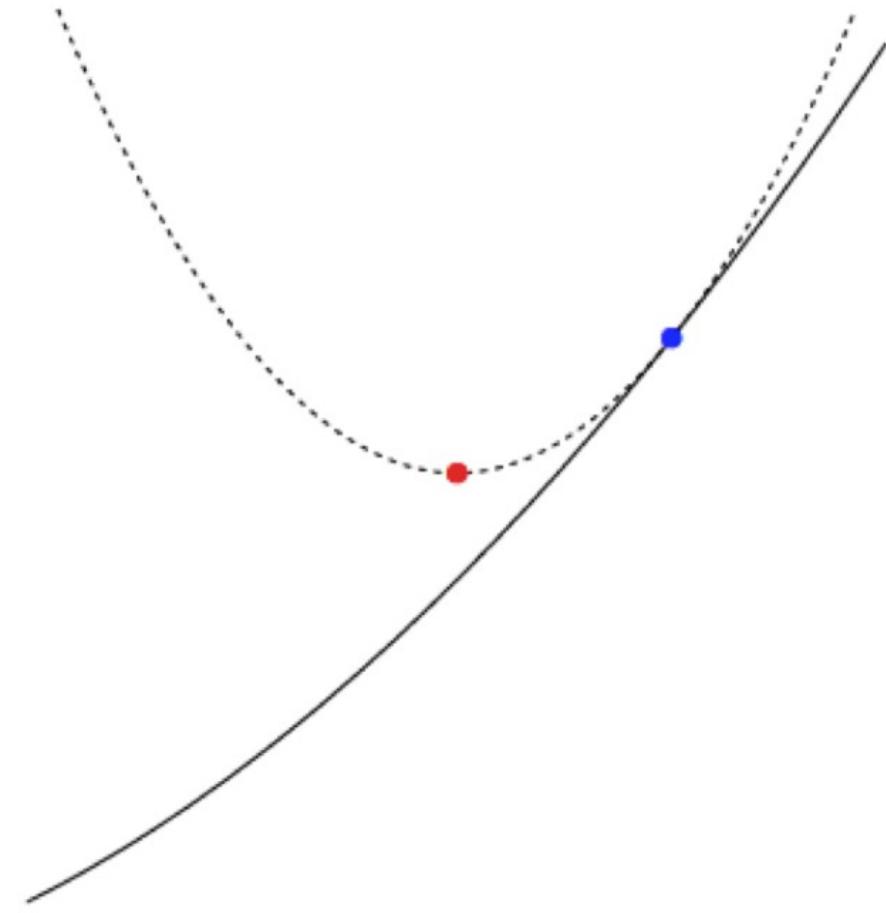
$$\mathbf{x}^+ = \arg \min_{\mathbf{y}} f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) + \frac{1}{2\alpha} \|\mathbf{y} - \mathbf{x}\|_2^2$$

- Ta giải phương trình đạo hàm $\nabla f(\mathbf{y}) = \mathbf{0}$.

$$\nabla f(\mathbf{x}) + \frac{1}{\alpha} (\mathbf{y} - \mathbf{x}) = \mathbf{0}$$

$$\mathbf{y} = \mathbf{x}^+ = \mathbf{x} - \alpha \nabla f(\mathbf{x})$$

- Nếu α nhỏ thì hệ số khoảng cách $\frac{1}{2\alpha}$ sẽ lớn và kích thước bước đi do đó sẽ nhỏ.
- Ngược lại, nếu α lớn thì hệ số khoảng cách $\frac{1}{2\alpha}$ sẽ nhỏ và kích thước bước đi có thể lớn. Hệ số học (learning rate) α còn được gọi là kích thước bước đi (step size).



Ví dụ

- Tại mỗi điểm x , ta hình thành một xấp xỉ bậc hai của hàm số f cần cực tiểu hóa tại x :

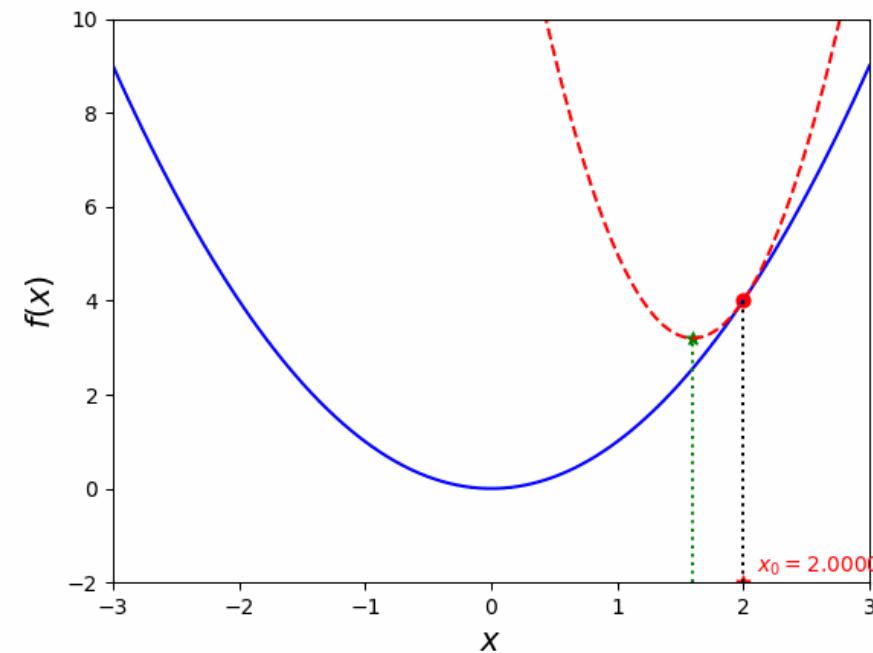
$$f(y) \approx f(x) + \nabla f(x) \cdot (y - x) + \frac{1}{2\alpha} \|y - x\|_2^2$$

- Gradient Descent chọn điểm kế tiếp $y = x^+$ sao cho cực tiểu hóa được xấp xỉ bậc hai này:

$$x^+ = \arg \min_y f(x) + \nabla f(x) \cdot (y - x) + \frac{1}{2\alpha} \|y - x\|_2^2$$

- Một bước của Gradient Descent chính là:

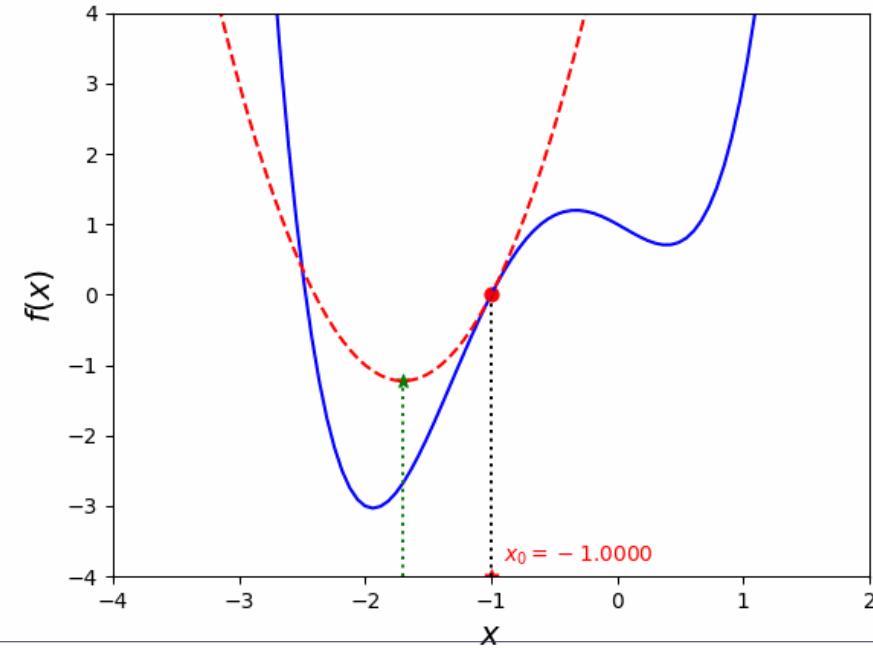
$$y = x^+ = x - \alpha \nabla f(x)$$





Ví dụ

- Nếu hàm số f không phải là hàm lồi (non-convex) thì tồn tại nhiều **cực tiểu địa phương** (local minima). Lời giải trả về bởi Gradient Descent **phụ thuộc** vào **điểm bắt đầu** x_0 .
- Ta có thể thu được lời giải **cực tiểu toàn cục** (global minimum).
- Hoặc một lời giải **cực tiểu địa phương** (local minimum) mà không phải là **cực tiểu toàn cục**.
- Nếu hệ số học quá lớn, thì thuật toán Gradient Descent có thể không hội tụ được.



```
def f(x):  
    return x**4 + 2.5*(x**3) - 0.5*(x**2) - x + 1  
  
def gradient(x):  
    return 4.0*(x**3) + 3.0*2.5*(x**2) - x - 1.0  
  
def quadratic_app(f, x, y, gradient, alpha):  
    return f(x) + gradient(x)*(y-x) + 1.0/(2.0*alpha) * (y-x)*(y-x)
```