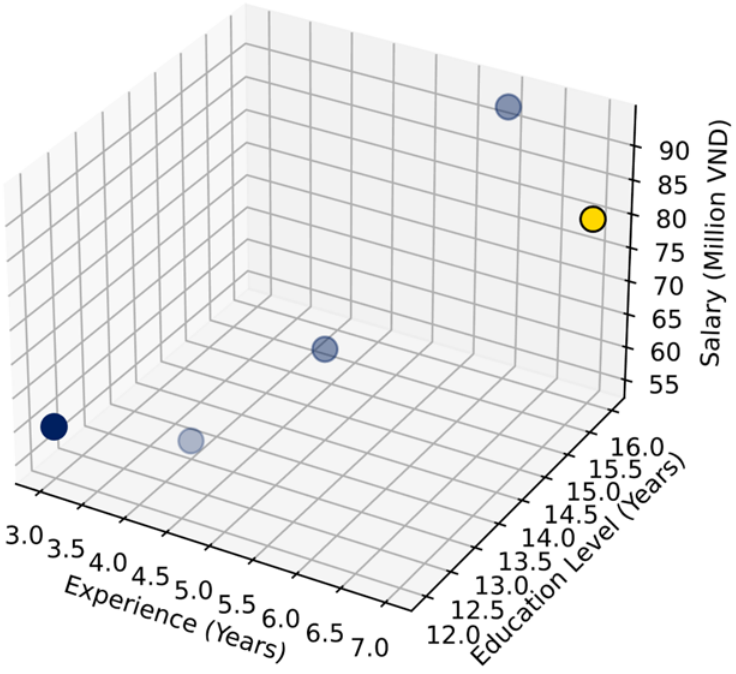
AI VIET NAM – AI COURSE 2024

Step-by-Step Vectorized Linear Regression

Yen-Linh Vu, Dinh-Thang Duong, Quang-Vinh Dinh

I. Dẫn nhập

Trong Simple Linear Regression, chúng ta đã biết về bài toán dự đoán tiền lương của nhân viên dựa trên số năm kinh nghiệm của họ. Song, một nhược điểm trong cách triển khai đơn giản này nằm ở việc ta cần triển khai thủ công rất nhiều các phép tính toán trong cài đặt code khi số lượng tham số tăng lên. Đồng nghĩa rằng, giả sử với d đặc trưng đầu vào (số cột), ta cần sử dụng các vòng lặp for để thực hiện việc tính toán tương ứng theo đúng các công thức trong Linear Regression. Điều này dẫn đến sự không tối ưu về mặt tính toán.



Experience	Education	Salary
3	12	60
4	13	55
5	14	66
6	15	93
7	16	?

Hình 1: Minh họa về bộ dữ liệu có 3 biến được trên không gian 3D, thể hiện số năm kinh nghiệm, trình độ học vấn ứng với tiền lương.

Để làm rõ vấn đề trên, ta xét một phương trình tuyến tính khi chỉ có một đặc trưng với tham số w, phương trình trở thành:

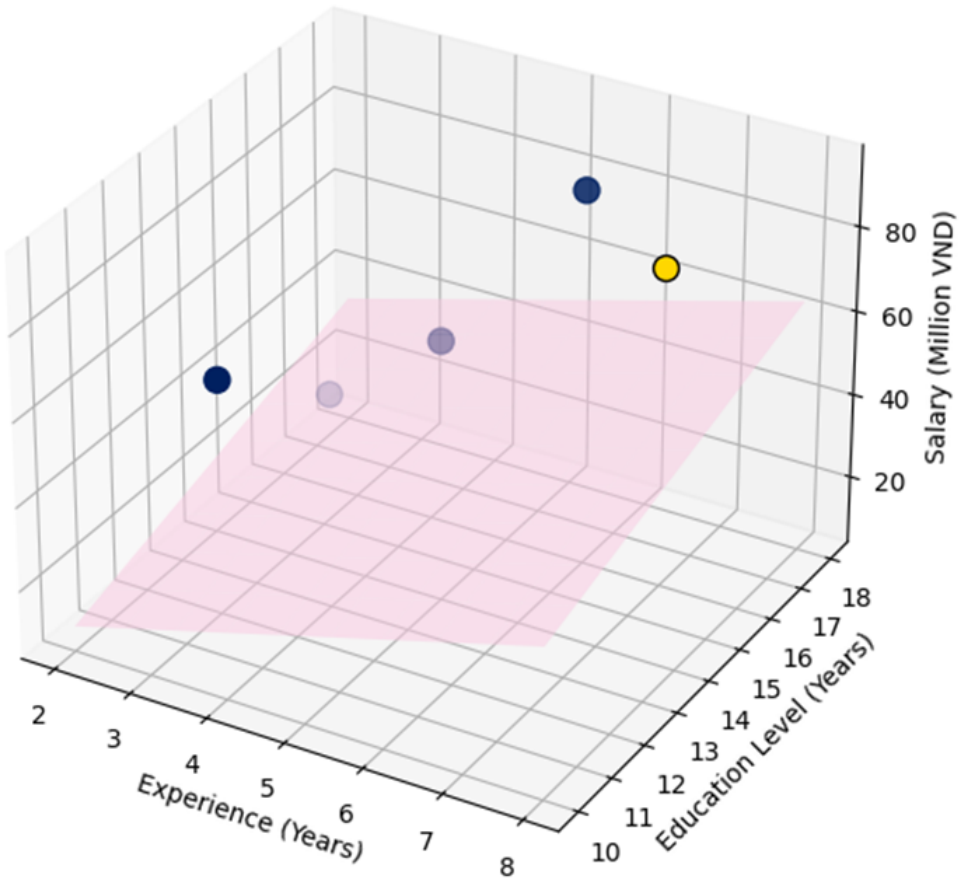
$$y = b + wx$$

Khi tăng số đặc trưng thành hai, tương ứng với các tham số w_1, w_2, phương trình trở thành:

$$y = b + w_1 x_1 + w_2 x_2$$

Khi có hai đặc trưng, đường thẳng dự đoán trong Linear Regression của chúng ta trở thành một mặt phẳng (plane) như hình 1. Và cứ như vậy, khi có d đặc trưng, ta sẽ suy ra phương trình tổng quát là:

$$y = b + \sum_{i=1}^{d} w_i x_i \tag{1}$$



Hình 2: Phương trình dự đoán trong Linear Regression (lúc này là một mặt phẳng) trên không gian 3D khi bài toán của chúng ta có hai đặc trưng và một nhãn dữ liệu.

Có thể quan sát được rằng, trong bản cài đặt Simple Linear Regression, khi số lượng tham số tăng lên, ta cần sử dụng vòng lặp for trong lập trình Python cơ bản để có thể tính toán được công thức dự đoán output của bài toán. Cách triển khai như vậy cho thấy sự thiếu tối ưu dẫn đến thời gian thực thi chậm và việc cài đặt sẽ gặp khó khăn trong trường hợp số lượng đặc trưng đầu vào lớn. Vậy câu hỏi đặt ra là, làm cách nào để ta có thể khắc phục vấn đề trên?

Một cách làm hiệu quả trong trường hợp này, đó là chúng ta có thể sử dụng **vector**. Một vector là một tập hợp các giá trị mà chúng ta có thể hình dung như một danh sách có thứ tự, trong đó mỗi giá trị là một thành phần của vector. Ví dụ, với hai vector $\mathbf{a}$ và $\mathbf{b}$, mỗi vector có ba phần tử, chúng ta đầu tiên có thể biểu diễn dưới dạng công thức như sau:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Mặc định khi nói đến vector mà không có ràng buộc nào khác, ta sẽ hiểu đây là các vector cột, tức các phần tử sẽ được viết từ trên xuống. Theo đại số tuyến tính, ta có thể cộng hai vector này bằng cách chỉ cần cộng từng phần tử tương ứng một cách đồng thời:

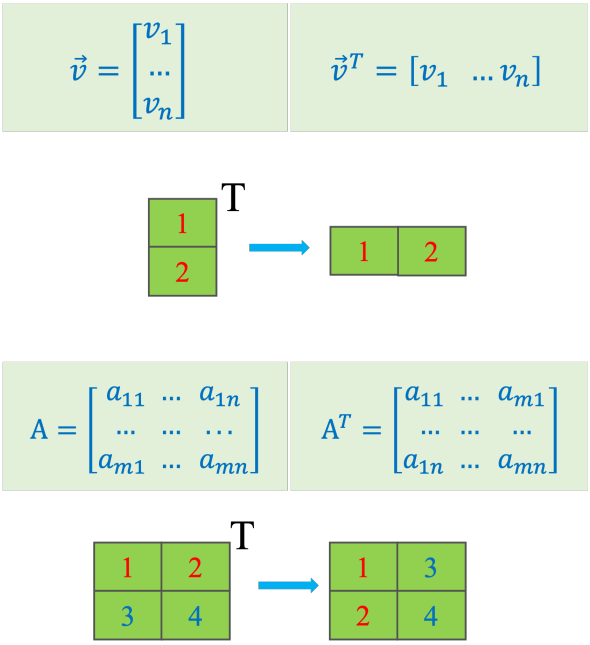
$$\mathbf{a} + \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ a_3 + b_3 \end{bmatrix}$$

Bên cạnh phép cộng, chúng ta còn có một phép toán rất quan trọng trong tính toán vector là phép chuyển vị (transpose). Nếu vector $\mathbf{a}$ ban đầu có dạng cột:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

thì phép chuyển vị của $\mathbf{a}$, ký hiệu là $\mathbf{a}^T$, sẽ biến nó thành một vector hàng:

$$\mathbf{a}^T = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}$$



Hình 3: Minh họa về phép chuyển vị với vector có kích thước $n \times 1$ và vector có kích thước $m \times n$.

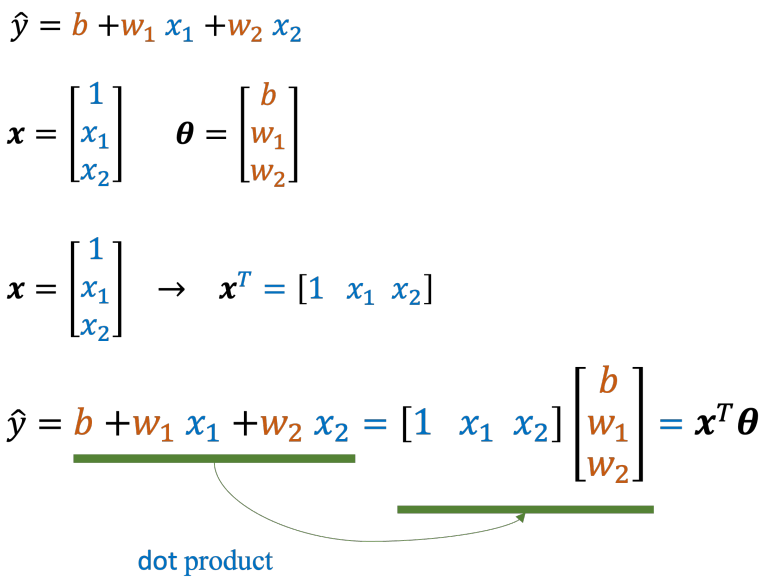
Cuối cùng, từ phép chuyển vị, chúng ta sẽ đến với phép tích vô hướng (dot product). Để hiểu rõ hơn, hãy xét hai vector **a** và **b** như trước, phép tích vô hướng giữa hai vector này sẽ được định nghĩa như sau:

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T\mathbf{b} = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = a_1b_1 + a_2b_2 + a_3b_3$$

Từ định nghĩa về vector và các phép tính có liên quan, chúng ta hoàn toàn có thể ứng dụng vào chương trình Linear Regression. Theo đó, các phép tính có thể được gộp lại vào trong vector để thành một lần thực thi duy nhất mà không cần vòng lặp. Thay vì xem từng tham số và đặc trưng riêng lẻ, chúng ta biểu diễn toàn bộ dãy tham số và đặc trưng dưới dạng vector. Điều này có nghĩa rằng, phương trình 1 sẽ được biểu đạt lại thành kết quả của phép tích vô hướng giữa vector tham số θ (bao gồm các trọng số w và bias b) và vector đặc trưng **x**:

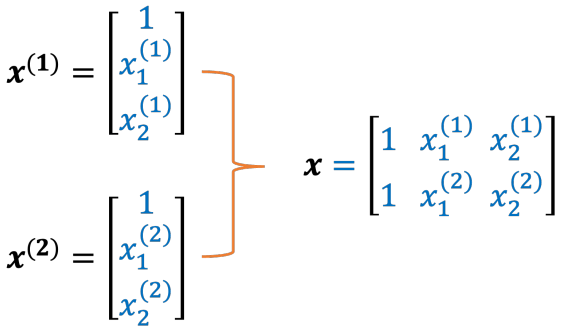
$$y = b + \sum_{i=1}^{d} w_i x_i = \mathbf{x} \cdot \theta = \mathbf{x}^T\theta = \begin{bmatrix} 1 & x_1 & x_2 & \dots & x_d \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \tag{2}$$

Với công thức tổng quát trên, giả sử số lượng đặc trưng là hai, phương trình tuyến tính được vector hóa sẽ có dạng như hình dưới đây:



Hình 4: Phương trình tuyến tính với 2 đặc trưng đầu vào phiên bản vector hóa.

Công thức trên đang giả định ta đang làm việc với một mẫu dữ liệu, vậy khi có N mẫu dữ liệu trong một bộ dữ liệu thì sẽ như thế nào? May mắn rằng, ta hoàn toàn có thể tiếp tục sử dụng phép tích vô hướng với N điểm dữ liệu đầu vào chỉ thông qua việc thay đổi một chút về cách bố trí vector dữ liệu đầu vào $\mathbf{x}$.



Hình 5: Minh họa vector dữ liệu đầu vào trong trường hợp có hai mẫu dữ liệu.

Theo đó, ta biểu diễn từng vector mẫu dữ liệu từ vector cột thành vector hàng, từ đó gom chúng lại vào một vector duy nhất. Ở hình số 5, với hai vector mẫu dữ liệu (kích thước 3×1), ta chuyển vị chúng thành vector hàng rồi gộp chung lại thành một vector có kích thước là 2×3. Khi đã gom lại thành một vector duy nhất, ta hoàn toàn có thể tính giá trị dự đoán cùng lúc cho cả hai mẫu dữ liệu này chỉ thông qua một phép tính tích vô hướng. Vẫn theo ví dụ trước đó, với hai mẫu dữ liệu, quá trình dự đoán của chúng ta sẽ diễn ra như hình sau:

$$\hat{y}^{(1)} = b + w_1 x_1^{(1)} + w_2 x_2^{(1)}$$

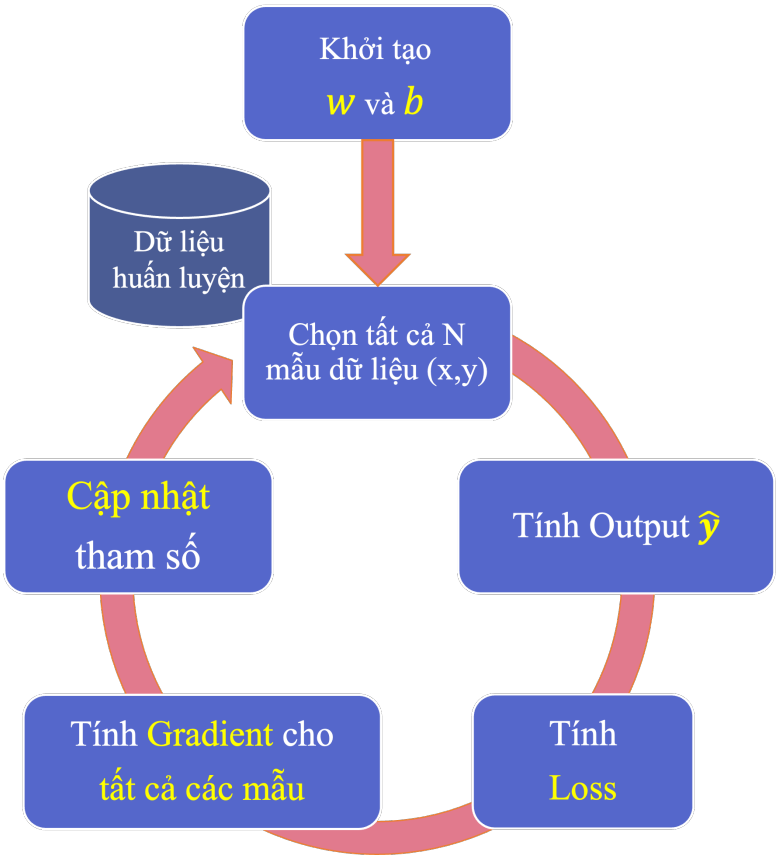
$$\hat{y}^{(2)} = b + w_1 x_1^{(2)} + w_2 x_2^{(2)}$$

$$\boldsymbol{x} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}$$

$$\hat{\boldsymbol{y}} = \boldsymbol{x\theta} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} b + w_1 x_1^{(1)} + w_2 x_2^{(1)} \\ b + w_1 x_1^{(2)} + w_2 x_2^{(2)} \end{bmatrix} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \end{bmatrix}$$

Hình 6: Minh họa phép tích vô hướng với trường hợp vector đầu vào có nhiều hơn một điểm dữ liệu. Việc tính toán vẫn chỉ diễn ra trong một phép toán duy nhất

Và đó chính là cách mà ta vector hóa việc dự đoán của hàm hồi quy tuyến tính. Với sự thay đổi này, việc tính toán cho các phần còn lại trong các bước tính toán của Gradient Descent cũng sẽ được cập nhật vector hóa tương ứng. Theo đó, chúng ta sẽ có một lược đồ thuật toán Gradient Descent ở dạng vector hóa và giải thích tương ứng như sau:



Hình 7: Pipeline tổng quát qua các bước tính toán để tối ưu hóa tham số θ (bao gồm cả trọng số w và bias b) sử dụng phương pháp **vectorization**.

Trong bài toán hồi quy tuyến tính đa biến truyền thống, chúng ta cần tối ưu hóa các trọng số w và bias b để giảm thiểu sai số giữa giá trị dự đoán $\hat{y}$ và giá trị thực tế y. Một cách tổng quát, với d trọng số, công thức hồi quy tuyến tính sẽ là:

$$\hat{y} = b + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \,,$$

trong đó:

- $x_1, x_2, \ldots, x_n$ là các đặc trưng đầu vào.
- $w_1, w_2, \ldots, w_n$ là các trọng số tương ứng với các đặc trưng.
- b là bias.
- $\hat{y}$ là giá trị dự đoán.

Tuy nhiên, khi áp dụng phương pháp **vectorization**, ta thay thế cách tính truyền thống bằng cách xử lý tất cả các mẫu dữ liệu cùng một lúc và tính toán đồng thời tất cả các trọng số w và bias b trong một phép tính tích vô hướng (dot product) duy nhất. Ta có phương trình tổng quát dạng vector hóa cho bài toán đa biến như sau:

$$\hat{\mathbf{y}} = \mathbf{X}\theta = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \ldots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \ldots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \ldots & x_d^{(N)} \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} ,$$

trong đó:

- $\mathbf{X}$ là ma trận dữ liệu đầu vào (kích thước $N \times d$), với d là số lượng đặc trưng (số cột không tính cột nhãn) và bias.
  - Ma trận $\mathbf{X}$ bao gồm N hàng, mỗi hàng đại diện cho một sample (mẫu dữ liệu). Mỗi cột của ma trận tương ứng với một đặc trưng (feature) đầu vào của mẫu. Cột đầu tiên của $\mathbf{X}$ chứa các giá trị bias, luôn bằng 1. Các cột tiếp theo chứa giá trị của các đặc trưng tương ứng cho mỗi sample, từ x_1 đến x_d.
- θ là vector tham số, chứa trọng số w và bias b (kích thước $d \times 1$), với d là số lượng đặc trưng (số cột không tính cột nhãn) và bias.
- $\hat{\mathbf{y}}$ là vector dự đoán (kích thước $N \times 1$).
- $\mathbf{y}$ là vector giá trị thực tế (kích thước $N \times 1$).

Khi bài toán được vector hóa, thuật toán Gradient Descent sẽ hoạt động theo các bước tổng quan như sau:

1. **Khởi tạo tham số ngẫu nhiên**: Khởi tạo vector tham số θ, bao gồm cả trọng số w và bias b. Tất cả các giá trị này ban đầu có thể được khởi tạo bằng 0 hoặc ngẫu nhiên.

2. Ở phiên bản Vectorized Linear Regression, ta sẽ cài đặt bổ sung một vòng lặp gọi là vòng lặp số epoch. Với ý tưởng rằng việc chỉ cho một mô hình học máy được nhìn qua các điểm dữ liệu một lần duy nhất sẽ không thể đạt được sự tối ưu ngay lập tức mà cần phải lặp đi lặp lại quy trình tính toán một cơ số lần. Chính vì lẽ đó, trong trường hợp này, ta sử dụng số epoch để đại diện cho số lần mà mô hình Linear Regression được "học"qua bộ dữ liệu. Như vậy, với epoch thứ *idx* trong vòng lặp số epoch, ta thực hiện các bước sau:

   (a) **Lấy tất cả các mẫu dữ liệu**:
   - $\mathbf{X}$ là ma trận đầu vào của tất cả các mẫu dữ liệu.
   - $\mathbf{y}$ là vector các giá trị thực tế.

   (b) **Tính vector giá trị dự đoán $\hat{\mathbf{y}}$:** Dự đoán giá trị đầu ra cho tất cả các mẫu dựa trên công thức vector hóa:
   $$\hat{\mathbf{y}} = \mathbf{X}\theta$$

   (c) **Tính Loss:** Hàm loss (khoảng cách giữa các điểm dữ liệu thực tế so với các điểm dự đoán tương ứng) được sử dụng để đo lường sự khác biệt giữa vector giá trị dự đoán $\hat{\mathbf{y}}$ và vector giá trị thực tế $\mathbf{y}$:
   $$\mathbf{L}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N}(\hat{\mathbf{y}} - \mathbf{y})^T \cdot (\hat{\mathbf{y}} - \mathbf{y})$$

   Trong đó, $\frac{1}{N}$ tính trung bình cho tất cả các mẫu, và $(\hat{\mathbf{y}} - \mathbf{y})^T$ biểu diễn vector dự đoán trừ đi giá trị thực, sau đó nhân với chính nó thông qua tích vô hướng (dot product). Khi áp dụng vectorization, ta có thể phân tích như sau:

   $$\begin{aligned}\mathbf{L}(\hat{\mathbf{y}}, \mathbf{y}) &= \frac{1}{N}\begin{bmatrix}(\hat{\mathbf{y}}^{(\mathbf{1})} - \mathbf{y}^{(\mathbf{1})}) & (\hat{\mathbf{y}}^{(\mathbf{2})} - \mathbf{y}^{(\mathbf{2})}) & \dots & (\hat{\mathbf{y}}^{(\mathbf{N})} - \mathbf{y}^{(\mathbf{N})})\end{bmatrix} \cdot \begin{bmatrix}(\hat{\mathbf{y}}^{(\mathbf{1})} - \mathbf{y}^{(\mathbf{1})}) \\ (\hat{\mathbf{y}}^{(\mathbf{2})} - \mathbf{y}^{(\mathbf{2})}) \\ \vdots \\ (\hat{\mathbf{y}}^{(\mathbf{N})} - \mathbf{y}^{(\mathbf{N})})\end{bmatrix} \\ &= \frac{1}{N}(\hat{\mathbf{y}} - \mathbf{y})^T \cdot (\hat{\mathbf{y}} - \mathbf{y})\end{aligned}$$

   Các chỉ số (i) đại diện cho sample thứ i trong tổng số N samples của tập dữ liệu. Nói cách khác:
   - $\hat{y}^{(i)}$ là giá trị dự đoán của sample thứ i, và $y^{(i)}$ là giá trị thực tế tương ứng với sample thứ i, với $i = 1, 2, \dots, N$.

   (d) **Tính đạo hàm:** Ta đã biết rằng, trong phiên bản cài đặt Simple Linear Regression với d số đặc trưng, công thức tính đạo hàm cho tham số w và b có một chút sự khác biệt. Trong trường hợp có N mẫu dữ liệu, ta có các công thức tương ứng:
   - Đạo hàm hàm loss theo tham số w_j:
   $$\frac{\partial L}{\partial w_j} = \frac{2}{N}\sum_{i=1}^{N} x_{ij}(\hat{y}_i - y_i)$$

- Đạo hàm hàm loss theo bias b:

$$\frac{\partial L}{\partial b} = \frac{2}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)$$

Song, sự khác biệt chỉ nằm ở việc đạo hàm của tham số w cần có sự tham gia của giá trị đầu vào x. Vì vậy, ta vẫn có thể tạo ra một công thức tổng quát để phục vụ cho việc vector hóa diễn ra thuận lợi. Theo đó, nhận thấy phần chung trong công thức đạo hàm của hai tham số là $(\hat{y}_i - y_i)$, ta tạm gán phép tính này trong trường hợp đã vector hóa vào một vector $\mathbf{k}$:

$$\mathbf{k} = 2(\hat{\mathbf{y}} - \mathbf{y})$$

Nhận thấy rằng, trong ma trận dữ liệu đầu vào đã cung cấp sẵn thông tin rằng bias sẽ không có phần tử x, còn với các đặc trưng thứ i sẽ có các phần tử x_i tương ứng đi kèm. Vậy nên, nhiệm vụ lúc này của ta chỉ việc nhân tương ứng ma trận $\mathbf{X}$ và vector $\mathbf{k}$ lại thì việc tính đạo hàm ở dạng vector hóa đã được hoàn tất:

$$\mathbf{L}'_\theta = \mathbf{X}^T\mathbf{k} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{11} & x_{21} & \dots & x_{N1} \\ x_{12} & x_{22} & \dots & x_{N2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1d} & x_{2d} & \dots & x_{Nd} \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_N \end{bmatrix} = \begin{bmatrix} k_1 + k_2 + \dots + k_n \\ k_1x_{11} + k_2x_{21} + \dots + k_nx_{n1} \\ k_1x_{12} + k_2x_{22} + \dots + k_nx_{n2} \\ \vdots \\ k_1x_{1d} + k_2x_{2d} + \dots + k_nx_{nd} \end{bmatrix}$$

(e) **Cập nhật tham số:** Với vector θ và vector các giá trị đạo hàm $\mathbf{L}'_\theta$, ta cập nhật tất cả các trọng số và bias cùng lúc thông qua công thức sau:

$$\theta = \theta - \eta\frac{\mathbf{L}'_\theta}{N}$$

Trong đó η là learning rate và N là số lượng mẫu.

3. **Lặp lại quá trình:** Quá trình trên được lặp lại cho đến khi hội tụ, tức là hàm loss đạt đến giá trị nhỏ nhất hoặc thay đổi của các tham số rất nhỏ.

II. Thực hành

Dựa theo cách triển khai đã đề cập ở phía trên, chúng ta sẽ thử áp dụng để xây dựng một hàm có khả năng dự đoán tiền lương của nhân viên dựa trên số năm kinh nghiệm làm việc và trình độ học vấn của người đó. Đầu tiên, chúng ta xem qua một bộ dữ liệu nhỏ theo bảng dưới đây:

Index	Experience	Education	Salary (.million VND)
0	3	12	60
1	4	13	55
2	5	14	66
3	6	15	93

Bảng 1: Bộ dữ liệu về thông tin tiền lương của nhân viên theo số năm kinh nghiệm và trình độ học vấn tại công ty.

Bộ dữ liệu trên gồm có 4 mẫu (sample), ứng với số năm kinh nghiệm (Experience) và trình độ học vấn (Education) sẽ có số tiền lương tương ứng (Salary). Ví dụ, tại mẫu dữ liệu có Index bằng 1, một nhân viên có 4 năm kinh nghiệm và 13 năm học vấn sẽ có tiền lương tương ứng là 55 triệu VND. Với đề bài trên, ta sẽ tiến hành áp dụng thuật toán mô tả ở phần Dẫn nhập theo từng bước như sau:

1. **Khởi tạo giá trị ban đầu cho các tham số w_1, w_2, và b:**
   Ở đây, ta giả sử $w_1 = 3$, $w_2 = 2$, và $b = 10$. Chúng ta cũng sẽ chọn giá trị $\eta = 0.001$ để sử dụng cho phần cập nhật tham số và số *epoch* $= 2$ cho quá trình lặp qua bộ dữ liệu của thuật toán.

   Với các giá trị này, ta thử áp dụng công thức dự đoán và tính **loss** để xem mức độ hiệu quả. Đầu tiên, ta sẽ dự đoán mức lương của một nhân viên có 7 năm kinh nghiệm và 16 năm học vấn với công thức sau:

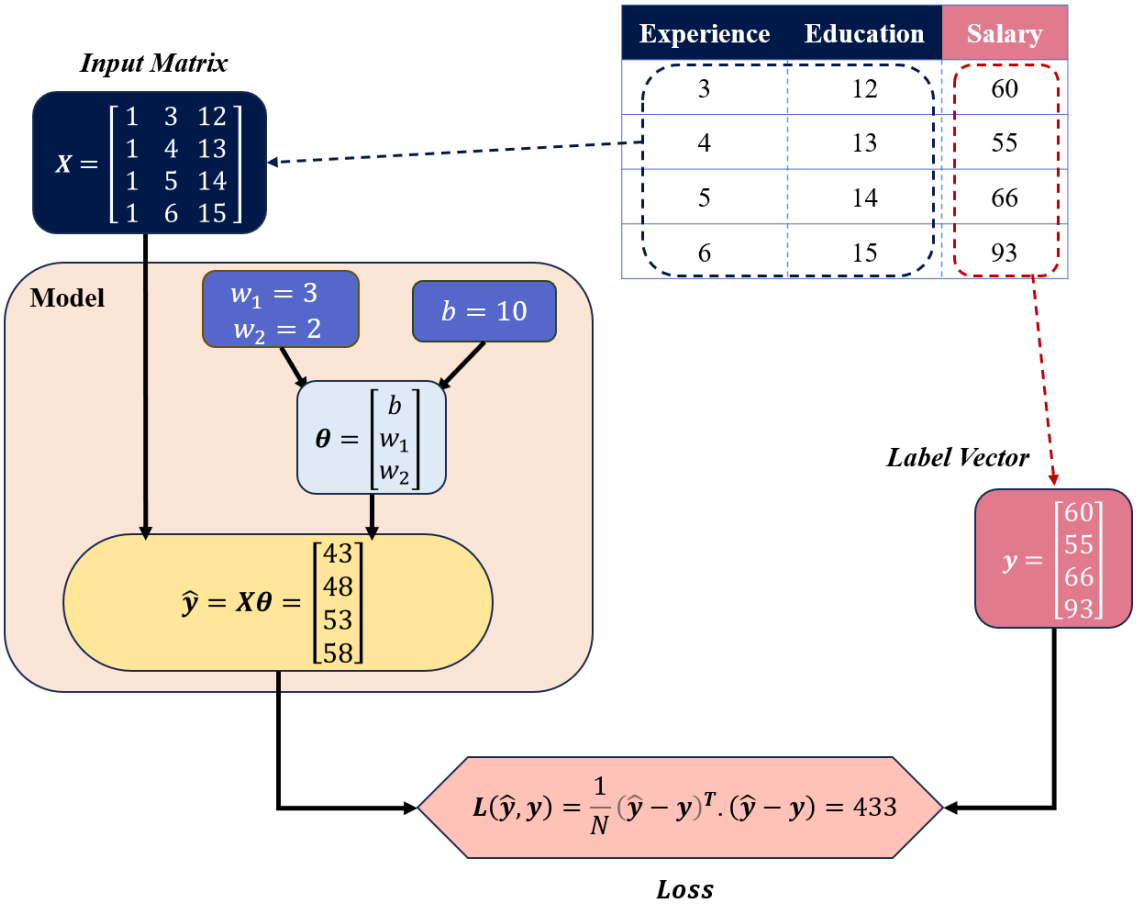
$$\hat{y} = [1 \quad x_1 \quad x_2] \cdot \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix} = 10 + 3 \times 7 + 2 \times 16 = 63$$

   Tiếp theo, ta sẽ tính toán **loss** giữa giá trị dự đoán và giá trị thực tế (giả định là **110 triệu VND**), do hiện tại chỉ xét trên 1 mẫu dữ liệu (N=1), ta sử dụng công thức đơn giản hóa $L(\hat{y}, y) = (\hat{y} - y)^2$ mà không cần chia cho N:

$$L(\hat{y}, y) = (\hat{y} - y)^2 \ = (63 - 110)^2 = 2209$$

   Với giá trị **loss** này, ta có thể thấy rằng cần tiếp tục điều chỉnh các tham số w_1, w_2, và b để cải thiện độ chính xác của mô hình.

2. Với *epoch* thứ 1 trong vòng lặp số epoch, ta thực hiện các bước sau:



Hình 8: Minh họa quá trình Forward Propagation, bao gồm lấy dữ liệu (a), tính $\hat{\mathbf{y}}$ (b) và tính Loss (c).

(a) **Lấy tất cả các mẫu dữ liệu:**
Ma trận đầu vào $\mathbf{X}$, giá trị thực tế $\mathbf{y}$, tham số khởi tạo $\theta = [b, w_1, w_2]$, learning rate η, và số lượng mẫu N:

$$\mathbf{X} = \begin{bmatrix} 1 & 3 & 12 \\ 1 & 4 & 13 \\ 1 & 5 & 14 \\ 1 & 6 & 15 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 60 \\ 55 \\ 66 \\ 93 \end{bmatrix}, \quad \theta = \begin{bmatrix} 10 \\ 3 \\ 2 \end{bmatrix}, \quad \eta = 0.001, \quad N = 4$$

(b) **Tính giá trị dự đoán ŷ:**
Giá trị dự đoán $\hat{\mathbf{y}}$ được tính dựa trên ma trận đầu vào $\mathbf{X}$ và tham số θ:

$$\hat{\mathbf{y}} = \mathbf{X}\theta = \begin{bmatrix} 1 & 3 & 12 \\ 1 & 4 & 13 \\ 1 & 5 & 14 \\ 1 & 6 & 15 \end{bmatrix} \begin{bmatrix} 10 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 43 \\ 48 \\ 53 \\ 58 \end{bmatrix}$$

(c) **Tính Loss:**

Hàm loss được sử dụng để đo lường sự khác biệt giữa giá trị dự đoán $\hat{\mathbf{y}}$ và giá trị thực tế $\mathbf{y}$:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N}(\hat{\mathbf{y}} - \mathbf{y})^T \cdot (\hat{\mathbf{y}} - \mathbf{y})$$

Để tính Loss, trước hết chúng ta cần tính **sai số** giữa giá trị dự đoán $\hat{\mathbf{y}}$ và giá trị thực tế $\mathbf{y}$.

$$\hat{\mathbf{y}} - \mathbf{y} = \begin{bmatrix} 43 \\ 48 \\ 53 \\ 58 \end{bmatrix} - \begin{bmatrix} 60 \\ 55 \\ 66 \\ 93 \end{bmatrix} = \begin{bmatrix} -17 \\ -7 \\ -13 \\ -35 \end{bmatrix}$$

Với $N = 4$, ta có:

$$L(\hat{y}, y) = \frac{1}{4}(\hat{y}-y)^T \cdot (\hat{y}-y) = \frac{1}{4}\begin{bmatrix} -17 & -7 & -13 & -35 \end{bmatrix} \cdot \begin{bmatrix} -17 \\ -7 \\ -13 \\ -35 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 289 & 49 & 169 & 1225 \end{bmatrix}$$

$$= \frac{1}{4} \times (289 + 49 + 169 + 1225) = \frac{1}{4} \times 1732 = 433$$

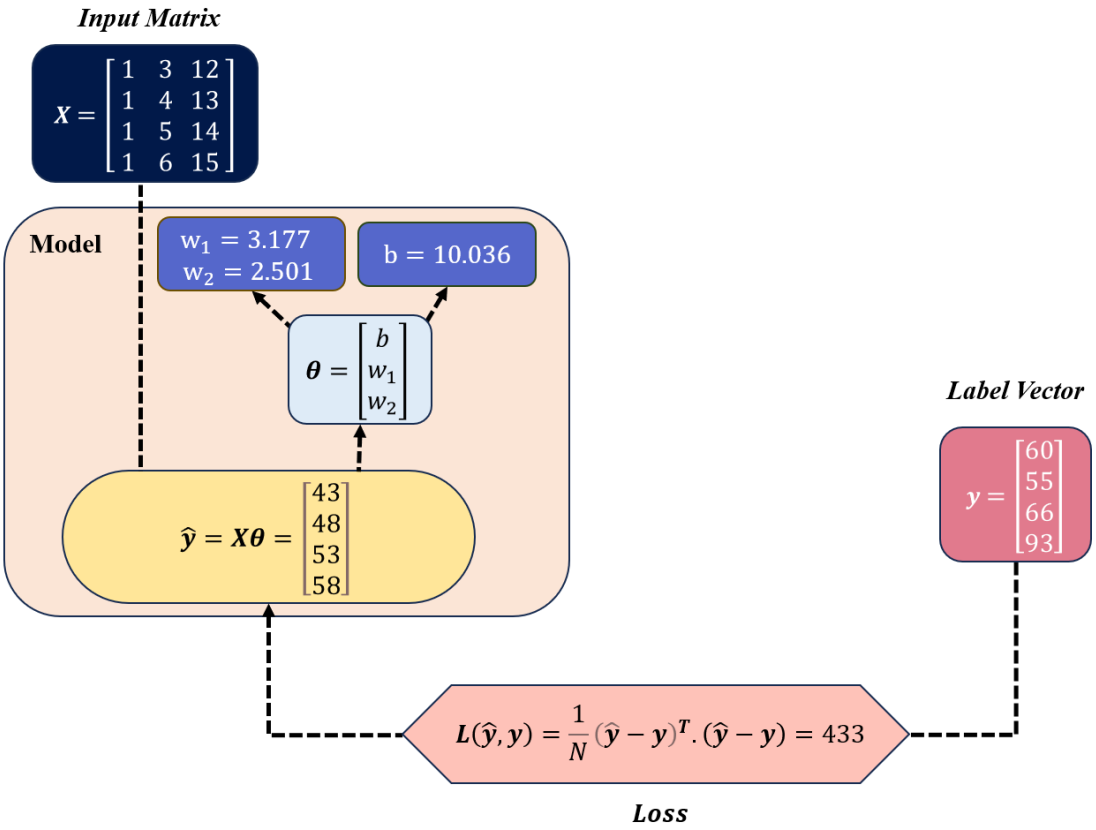
Hàm loss trên được thiết kế nhằm đơn giản hóa việc cài đặt. Khi sử dụng numpy để cài đặt, chúng ta sẽ ánh xạ chính xác công thức được mô tả vào code numpy.

Trong một số tài liệu, khi mô tả hàm loss về mặt lý thuyết (không quan tâm đến khía cạnh cài đặt), hàm loss thường được biểu diễn như sau:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})^2$$

Như vậy, Loss cũng có thể tính như dưới đây:

$$\begin{aligned} L(\hat{\mathbf{y}}, \mathbf{y}) &= \frac{1}{4}\left[(-17)^2 + (-7)^2 + (-13)^2 + (-35)^2\right] \\ &= \frac{1}{4} \times (289 + 49 + 169 + 1225) \\ &= \frac{1}{4} \times 1732 \\ &= 433 \end{aligned}$$



Hình 9: Quá trình Backpropagation, bao gồm tính Gradient(d) và cập nhật tham số (e).

(d) **Tính Gradient:**

Đạo hàm của hàm mất mát theo giá trị dự đoán $\hat{\mathbf{y}}$:

$$\mathbf{k} = 2 \times (\hat{\mathbf{y}} - \mathbf{y})$$

Áp dụng vào, ta có:

$$\mathbf{k} = 2 \times \begin{bmatrix} -17 \\ -7 \\ -13 \\ -35 \end{bmatrix} = \begin{bmatrix} -34 \\ -14 \\ -26 \\ -70 \end{bmatrix}$$

Gradient tổng hợp cho các tham số θ được tính như sau:

$$\mathbf{L_0'} = \mathbf{X}^T\mathbf{k}$$

Ta có:

$$\mathbf{L_0'} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 3 & 4 & 5 & 6 \\ 12 & 13 & 14 & 15 \end{bmatrix} \begin{bmatrix} -34 \\ -14 \\ -26 \\ -70 \end{bmatrix} = \begin{bmatrix} -144 \\ -708 \\ -2004 \end{bmatrix}$$

(e) **Cập nhật tham số:**

Cập nhật tất cả các trọng số θ và bias cùng lúc thông qua gradient descent:

$$\theta = \theta - \eta \frac{\mathbf{L}'_\theta}{N}$$

Áp dụng với $N = 4$, t $\eta = 0.001$:

$$\theta = \begin{bmatrix} 10 \\ 3 \\ 2 \end{bmatrix} - 0.001 \times \frac{1}{4} \begin{bmatrix} -144 \\ -708 \\ -2004 \end{bmatrix} = \begin{bmatrix} 10 \\ 3 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.036 \\ 0.177 \\ 0.501 \end{bmatrix} = \begin{bmatrix} 10.036 \\ 3.177 \\ 2.501 \end{bmatrix}$$

3. **Lặp lại quá trình:**

Sau khi hoàn thành *epoch* 1, các tham số θ đã được cập nhật với $b = 10.036$, $w_1 = 3.177$, $w_2 = 2.501$. Chúng ta tiếp tục thử với *epoch* thứ 2. Với *epoch* thứ 2 này, ta sẽ tiếp tục quá trình huấn luyện và cập nhật các tham số theo gradient descent dựa trên giá trị mới của θ.

(a) **Lấy tất cả các mẫu dữ liệu:**

Ma trận đầu vào $\mathbf{X}$, giá trị thực tế $\mathbf{y}$, và tham số θ sau epoch 1 :

$$\mathbf{X} = \begin{bmatrix} 1 & 3 & 12 \\ 1 & 4 & 13 \\ 1 & 5 & 14 \\ 1 & 6 & 15 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 60 \\ 55 \\ 66 \\ 93 \end{bmatrix}, \quad \theta = \begin{bmatrix} 10.036 \\ 3.177 \\ 2.501 \end{bmatrix}$$

(b) **Tính giá trị dự đoán $\hat{\mathbf{y}}$:**

$$\hat{\mathbf{y}} = \mathbf{X}\theta = \begin{bmatrix} 1 & 3 & 12 \\ 1 & 4 & 13 \\ 1 & 5 & 14 \\ 1 & 6 & 15 \end{bmatrix} \begin{bmatrix} 10.036 \\ 3.177 \\ 2.501 \end{bmatrix} = \begin{bmatrix} 49.579 \\ 55.257 \\ 60.935 \\ 66.613 \end{bmatrix}$$

(c) **Tính Loss:**

Sai số giữa $\hat{\mathbf{y}}$ và $\mathbf{y}$ là:

$$\hat{\mathbf{y}} - \mathbf{y} = \begin{bmatrix} 49.579 \\ 55.257 \\ 60.935 \\ 66.613 \end{bmatrix} - \begin{bmatrix} 60 \\ 55 \\ 66 \\ 93 \end{bmatrix} = \begin{bmatrix} -10.421 \\ 0.257 \\ -5.065 \\ -26.387 \end{bmatrix}$$

Do đó, Loss được tính như sau:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{4} \left[(-10.421)^2 + (0.257)^2 + (-5.065)^2 + (-26.387)^2 \right] \approx 207.65$$

(d) **Tính Gradient:**

Đạo hàm của hàm mất mát theo giá trị dự đoán $\hat{\mathbf{y}}$:

$$\mathbf{k} = 2 \times (\hat{\mathbf{y}} - \mathbf{y})$$

Áp dụng vào, ta có:

$$\mathbf{k} = 2 \times \begin{bmatrix} -10.421 \\ 0.257 \\ -5.065 \\ -26.387 \end{bmatrix} = \begin{bmatrix} -20.842 \\ 0.514 \\ -10.13 \\ -52.774 \end{bmatrix}$$

Gradient tổng hợp cho các tham số θ được tính như sau:

$$\mathbf{L_0'} = \mathbf{X}^T\mathbf{k}$$

Với $N = 4$, ta có:

$$\mathbf{L_0'} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 3 & 4 & 5 & 6 \\ 12 & 13 & 14 & 15 \end{bmatrix} \begin{bmatrix} -20.842 \\ 0.514 \\ -10.13 \\ -52.774 \end{bmatrix} = \begin{bmatrix} -83.232 \\ -427.764 \\ -1176.852 \end{bmatrix}$$

(e) **Cập nhật tham số:**
Cập nhật tất cả các trọng số θ và bias cùng lúc thông qua gradient descent với $\eta = 0.001$:

$$\theta = \begin{bmatrix} 10.036 \\ 3.177 \\ 2.501 \end{bmatrix} - 0.001 \times \frac{1}{4} \begin{bmatrix} -83.232 \\ -427.764 \\ -1176.852 \end{bmatrix} \approx \begin{pmatrix} 10.057 \\ 3.284 \\ 2.795 \end{pmatrix}$$

4. **Thử lại với bài toán dự đoán lương của nhân viên:**

Với các tham số mới cập nhật được sau *epoch* thứ 2, $b = 10.057$, $w_1 = 3.284$, và $w_2 = 2.795$, ta thử lại với bài toán dự đoán mức lương dự đoán của một nhân viên có 7 năm kinh nghiệm và 16 năm học vấn:

$$\hat{y} = 10.057 + 3.284 \times 7 + 2.795 \times 16 = 77.765 \text{ triệu VND}$$

Giá trị lương thực tế được giả định là 110 triệu VND. Ta tính lại Loss:

$$L(\hat{y}, y) = (77.765 - 110)^2 = 1039.095$$

Vậy là so với Loss ta tính được trước đó là 2209, giá trị Loss sau *epoch* 2 đã giảm đáng kể xuống còn 1039. Điều này cho thấy rằng việc sử dụng phương pháp **vectorization** trong quá trình huấn luyện với gradient descent đã giúp cải thiện độ chính xác của mô hình Linear Regression, làm giảm đáng kể sai số dự đoán qua các lần cập nhật tham số.

- Hết -