Module 6 - Tuần 2 - Softmax Regression

Time-Series Team

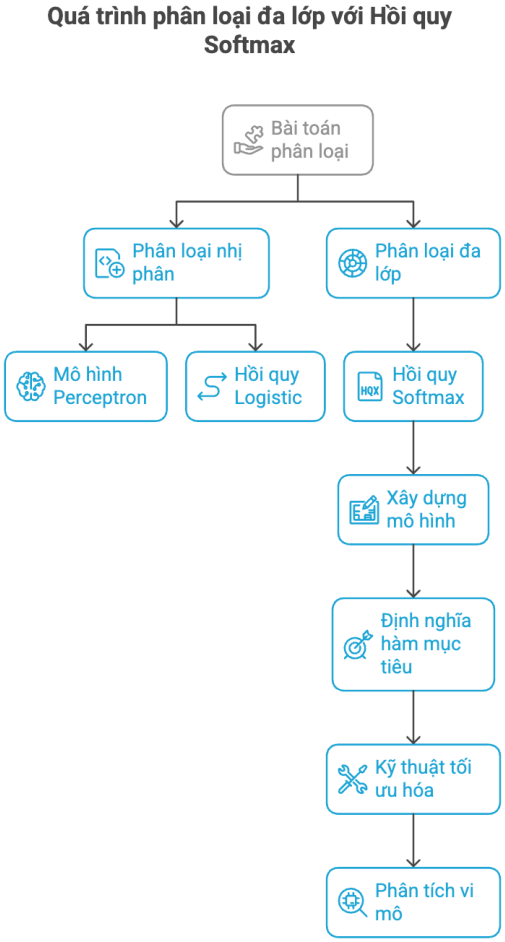
Ngày 12 tháng 11 năm 2025

Mở đầu và bối cảnh

Sự Chuyển Dịch Từ Phân Loại Nhị Phân Đến Đa Lớp

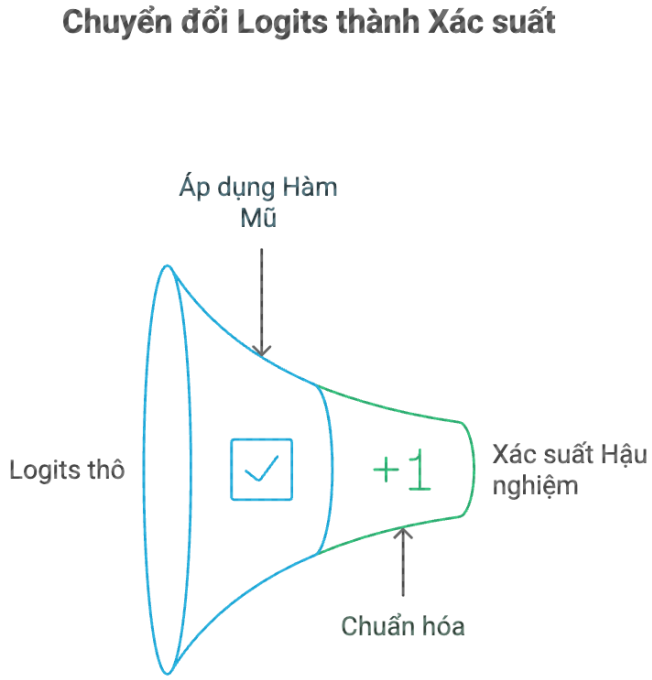
Trong bối cảnh phát triển của trí tuệ nhân tạo và học máy hiện đại, bài toán **phân loại (classification)** đóng vai trò trung tâm, là nền tảng cho vô số ứng dụng từ nhận diện hình ảnh y tế đến xử lý ngôn ngữ tự nhiên. Trong khi các mô hình sơ khai như **Perceptron** hay **Hồi quy Logistic (Logistic Regression)** đã giải quyết hiệu quả bài toán **phân loại nhị phân (binary classification)** - nơi không gian đầu ra chỉ giới hạn trong hai trạng thái đối ngẫu $\{0, 1\}$ - nhu cầu thực tế đòi hỏi khả năng xử lý các kịch bản phức tạp hơn với số lượng lớp $K > 2$.

Hồi quy Softmax, hay còn được gọi là **Hồi quy Logistic Đa thức (Multinomial Logistic Regression)**, không chỉ là một sự mở rộng đơn thuần về mặt kỹ thuật của **Hồi quy Logistic**, mà còn đại diện cho một bước nhảy vọt trong việc mô hình hóa phân phối xác suất trên các biến biến thiên rời rạc. Dựa trên tài liệu tham khảo 1 và 1, blog này sẽ tuân theo dòng chảy tư duy từ việc xây dựng mô hình, định nghĩa hàm mục tiêu, đến các kỹ thuật tối ưu hóa, nhưng sẽ đi sâu phân tích ở cấp độ vi mô của toán học và kỹ thuật tính toán.



Bản Chất Của Bài Toán Phân Loại Đa Lớp

Khác với bài toán hồi quy tuyến tính nơi đầu ra là một giá trị thực liên tục $y \in \mathbb{R}$, bài toán phân loại đa lớp yêu cầu mô hình phải đưa ra dự đoán thuộc về một tập hợp hữu hạn các lớp $C = \{1, 2, \ldots, K\}$. Một thách thức lớn đặt ra là làm thế nào để ánh xạ các đầu ra thô (logits) từ một mô hình tuyến tính – vốn có miền giá trị $(-\infty, +\infty)$ - sang một không gian xác suất hợp lệ, nơi tổng các thành phần bằng 1 và mỗi thành phần đều không âm.



Hàm **Softmax** giải quyết vấn đề này bằng cách tận dụng tính chất của hàm mũ (exponential function) để đảm bảo tính dương, và cơ chế chuẩn hóa (normalization) để đảm bảo tổng bằng đơn vị. Điều này cho phép chúng ta diễn giải đầu ra của mạng nơ-ron như là xác suất hậu nghiệm $P(y = k|x)$.1 Sự chuyển đổi này không chỉ có ý nghĩa về mặt toán học mà còn cung cấp khả năng diễn giải (interpretability) cho mô hình, cho phép các kỹ sư đánh giá độ tin cậy (confidence) của dự đoán.

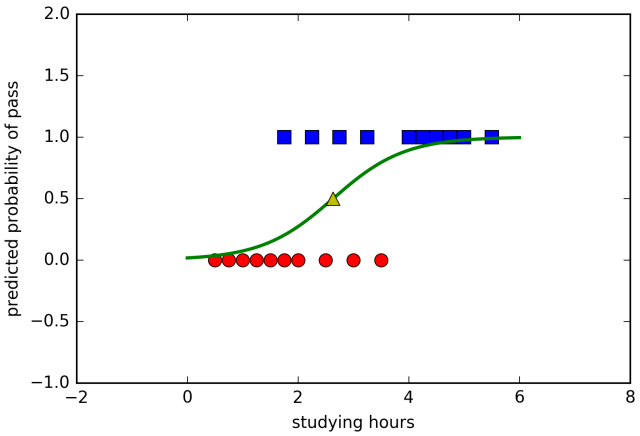
Mục lục

1 Mô Hình Hóa Toán Học: Từ Logistic Đến Softmax

1.1 Hồi Quy Logistic: Nền Tảng Của Phân Loại

Để hiểu sâu về **Softmax**, trước hết cần xem xét lại **Hồi quy Logistic**. Với bài toán phân loại nhị phân, ta mô hình hóa xác suất của lớp dương tính ($y = 1$) thông qua hàm **Sigmoid** $\sigma(z)$:

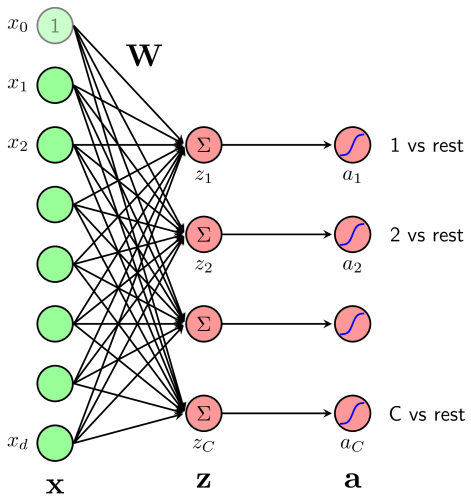
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

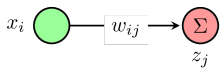


Trong đó $z = w^T x + b$ là tổ hợp tuyến tính của các đặc trưng đầu vào. **Hàm Sigmoid** nén z vào khoảng $(0, 1)$, và ta ngầm định rằng $P(y = 0|x) = 1 - P(y = 1|x)$. Điều này hoạt động tốt vì chỉ có một bậc tự do trong phân phối xác suất nhị phân. Tuy nhiên, khi số lớp K tăng lên, việc sử dụng C bộ phân loại nhị phân độc lập (chiến lược One-vs-Rest) sẽ gặp vấn đề về việc các xác suất không cộng lại bằng 1, gây khó khăn cho việc so sánh trực tiếp giữa các lớp.

1.2 Xây Dựng Mô Hình Softmax (Model Construction)

Trong Hồi quy Softmax, thay vì một đầu ra vô hướng, mô hình duy trì C đầu ra riêng biệt, tương ứng với điểm số (score) hoặc logit cho mỗi lớp.





w_{0j}: biases, don't forget!

d: data dimension

C: number of classes

$\mathbf{x} \in \mathbb{R}^{d+1}$

$\mathbf{W} \in \mathbb{R}^{(d+1)\times C}$

$z_i = \mathbf{w}_i^T \mathbf{x}$

$\mathbf{z} = \mathbf{W}^T \mathbf{x} \in \mathbb{R}^C$

$a_i = \text{sigmoid}(z_i) \in \mathbb{R}$

$0 < a_i < 1$

Giả sử dữ liệu đầu vào $x \in \mathbb{R}^d$ (hoặc $\mathbb{R}^n$), và có C lớp. Mô hình sẽ có một ma trận trọng số $W \in \mathbb{R}^{d \times C}$ và vector bias $b \in \mathbb{R}^C$. Quá trình tính toán xuôi (forward pass) bắt đầu bằng phép biến đổi affine:

$$z = W^T x + b$$

Ở đây, $z \in \mathbb{R}^C$ là vector logits. Thành phần z_i biểu thị mức độ "tự tin" chưa chuẩn hóa của mô hình đối với việc đầu vào x thuộc về lớp i. Giá trị z_i càng lớn, khả năng x thuộc lớp i càng cao. Tuy nhiên, z_i không bị chặn và không thể dùng trực tiếp làm xác suất.

1.3 Định Nghĩa Hàm Softmax

Hàm Softmax $\mathcal{S} : \mathbb{R}^C \to \mathbb{R}^C$ thực hiện phép chiếu vector logits z vào simplex xác suất $(C-1)$-chiều. Công thức cho phần tử thứ i của đầu ra Softmax là:

$$\hat{y}_i = \mathcal{S}(z_i) = \frac{e^{z_i}}{\sum j = 1^C e^{z_j}} \quad \text{với } i = 1, \ldots, C$$

Hàm này đảm bảo hai tính chất tiên quyết của phân phối xác suất:

1. **Dương tính (Positivity)**: $e^{z_i} > 0 \forall z_i \in \mathbb{R}$, do đó $\hat{y}_i > 0$.
2. **Chuẩn hóa (Normalization)**: $\sum_{i=1}^{C} \hat{y}_i = \frac{\sum_{i=1}^{C} e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} = 1$.

Mối Liên Hệ Với Vật Lý Thống Kê Một góc nhìn sâu sắc hơn về Softmax đến từ Vật lý thống kê, nơi hàm này tương đương với Phân phối Boltzmann (hoặc phân phối Gibbs). Trong ngữ cảnh đó, z_i đóng vai trò là năng lượng âm của một trạng thái ($E_i = -z_i$), và mẫu số $\sum e^{z_j}$ chính là hàm phân hoạch (partition function) Z. Nhiệt độ T thường được giả định bằng 1 trong học máy cơ bản, nhưng có thể được đưa vào dưới dạng $e^{z_i/T}$ để điều chỉnh độ "mềm" của phân phối – một kỹ thuật quan trọng trong Knowledge Distillation và Reinforcement Learning. (Theo Wikipedia)

1.4 So Sánh Cấu Trúc: One-hot Encoding

Để huấn luyện mô hình, ta cần so sánh đầu ra dự đoán $\hat{y}$ (vector xác suất) với nhãn thực tế y. Nhãn thực tế thường được cung cấp dưới dạng số nguyên (ví dụ: $y = 2$ cho lớp "mèo"). Để đồng bộ hóa không gian biểu diễn, ta sử dụng kỹ thuật One-hot Encoding.

Một vector one-hot $y \in \{0, 1\}^C$ được định nghĩa là vector có duy nhất một phần tử bằng 1 tại vị trí tương ứng với lớp đúng, và bằng 0 tại mọi vị trí khác. Ví dụ với $C = 3$:

Một vector one-hot $y \in \{0, 1\}^K$ được định nghĩa là vector có duy nhất một phần tử bằng 1 tại vị trí tương ứng với lớp đúng, và bằng 0 tại mọi vị trí khác.Ví dụ với $C = 3$:

- Lớp 0 $\to y = [1, 0, 0]^T$
- Lớp 1 $\to y = [0, 1, 0]^T$
- Lớp 2 $\to y = [0, 0, 1]^T$

Bảng dưới đây tóm tắt sự khác biệt về cấu trúc dữ liệu giữa Hồi quy Logistic và Softmax:

Bảng 1: So sánh Hồi quy Logistic và Hồi quy Softmax

Đặc Điểm	**Hồi Quy Logistic (Binary)**	**Hồi Quy Softmax (Multi-class)**
Số lượng lớp (C)	$C = 2$ (0 hoặc 1)	$C \geq 3$
Hàm kích hoạt	Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$	Softmax $\frac{e^{z_i}}{\sum e^{z_j}}$
Đầu ra mô hình	Một giá trị vô hướng $\hat{y} \in (0, 1)$	Vector $\hat{y} \in (0, 1)^C$, tổng bằng 1
Biểu diễn nhãn	Số vô hướng $y \in \{0, 1\}$	One-hot Vector $y \in \{0, 1\}^C$
Hàm mất mát	Binary Cross-Entropy	Categorical Cross-Entropy
Số tham số	$d + 1$ (1 vector trọng số + bias)	$C(d + 1)$ (ma trận trọng số + bias vector)

2 Phân Tích Tính Ổn Định Số Học (Numerical Stability)

Trong lý thuyết toán học thuần túy, hàm **Softmax** hoạt động hoàn hảo với mọi giá trị $z \in \mathbb{R}$. Tuy nhiên, trong môi trường tính toán số (như máy tính sử dụng chuẩn IEEE 754), việc cài đặt trực tiếp công thức $\frac{e^{z_i}}{\sum e^{z_j}}$ tiềm ẩn những rủi ro nghiêm trọng về độ chính xác và khả năng thực thi. Đây là một chủ đề được nhấn mạnh trong 1 và các tài liệu kỹ thuật.

2.1 Vấn Đề Tràn Số (Overflow) và Dưới Tràn (Underflow)

Hàm mũ e^x tăng trưởng cực kỳ nhanh.

- **Overflow**: Nếu một phần tử z_i có giá trị lớn (ví dụ $z_i = 1000$), e^{1000} sẽ vượt quá giới hạn biểu diễn của kiểu dữ liệu `float64` (giới hạn khoảng 1.8×10^{308}). Máy tính sẽ trả về `inf` (∞ - vô cùng). Khi đó, cả tử số và mẫu số đều là `inf`, dẫn đến kết quả `nan` (Not a Number) cho phép chia. Điều này làm hỏng toàn bộ quá trình huấn luyện.
- **Underflow**: Nếu z_i là một số âm có trị tuyệt đối lớn (ví dụ $z_i = -1000$), e^{z_i} sẽ bị làm tròn về 0. Nếu tất cả các z_i đều rất nhỏ, mẫu số $\sum e^{z_j}$ sẽ tiến về 0, dẫn đến lỗi chia cho 0 (ZeroDivisionError) hoặc kết quả không xác định.

2.2 Giải Pháp: Softmax Ổn Định (Stable Softmax)

Để khắc phục, ta sử dụng tính chất **bất biến với phép tịnh tiến** (translation invariance) của hàm **Softmax**. Nếu ta cộng hoặc trừ một hằng số C vào tất cả các phần tử của vector đầu vào z, phân phối xác suất đầu ra không thay đổi.

Chứng minh:

Giả sử $z' = z - C$ (trừ hằng số C vào mỗi phần tử). Khi đó:

$$\mathcal{S}(z')i = \frac{e^{z_i - C}}{\sum j = 1^K e^{z_j - C}} = \frac{e^{z_i} \cdot e^{-C}}{\sum_{j=1}^K (e^{z_j} \cdot e^{-C})} = \frac{e^{z_i} \cdot e^{-C}}{e^{-C} \cdot \sum_{j=1}^K e^{z_j}} = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} = \mathcal{S}(z)_i$$

Kỹ thuật tiêu chuẩn ("The Max Trick") là chọn $C = \max(z)$. Khi đó, vector logits mới z' sẽ có:

- Phần tử lớn nhất bằng 0 ($z_{max} - z_{max} = 0$), dẫn đến $e^0 = 1$.
- Tất cả các phần tử còn lại đều ≤ 0, dẫn đến $e^{z'_j} \in (0, 1]$.

Hệ quả:

1. Tử số và mẫu số luôn chứa ít nhất một phần tử bằng 1 (từ phần tử lớn nhất), do đó mẫu số luôn ≥ 1, loại bỏ hoàn toàn khả năng chia cho 0 (Underflow mẫu số).
2. Không có giá trị mũ nào vượt quá $e^0 = 1$, loại bỏ hoàn toàn khả năng tràn số dương (Overflow).
3. Các giá trị z_j rất nhỏ sẽ có $e^{z'_j} \approx 0$, nhưng điều này chấp nhận được vì chúng không đóng góp đáng kể vào tổng và xác suất đầu ra sẽ xấp xỉ 0 một cách chính xác.

2.3 LogSumExp Trick

Khi tính toán log-likelihood (thường dùng trong hàm mất mát), ta cần tính $\log(\sum e^{z_j})$. Việc tính tổng mũ rồi mới lấy logarit cũng gặp vấn đề số học tương tự. Kỹ thuật "LogSumExp" kết hợp phép trừ max vào trong quá trình tính toán log:

$$\log\left(\sum_j e^{z_j}\right) = \log\left(\sum_j e^{z_j - C} \cdot e^C\right) = \log\left(e^C \cdot \sum_j e^{z_j - C}\right) = C + \log\left(\sum_j e^{z_j - C}\right)$$

Với $C = \max(z)$, công thức này đảm bảo sự ổn định tuyệt đối khi tính hàm mất mát.

Đây là lý do tại sao các thư viện như PyTorch cung cấp hàm `LogSoftmax` hoặc tích hợp sẵn trong `CrossEntropyLoss` thay vì khuyến khích người dùng tự tính Softmax rồi Log.

3 Hàm Mất Mát Cross-Entropy (Cross-Entropy Loss)

3.1 Từ Nguyên Lý Ước Lượng Hợp Lý Cực Đại (MLE)

Mục tiêu của việc huấn luyện mô hình là tìm ra bộ tham số $\theta = \{W, b\}$ sao cho xác suất mô hình gán cho nhãn đúng là cao nhất. Giả sử tập dữ liệu huấn luyện gồm m mẫu độc lập $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$. Hàm hợp lý (Likelihood function) được định nghĩa là:

$$\mathcal{L}(\theta) = \prod_{i=1}^{m} P(y^{(i)}|x^{(i)}; \theta)$$

Để thuận tiện cho việc tối ưu hóa (biến phép nhân thành phép cộng) và tránh underflow số học khi nhân nhiều xác suất nhỏ, ta tối đa hóa hàm Log-Likelihood:

$$\ell(\theta) = \log \mathcal{L}(\theta) = \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}; \theta)$$

Trong bài toán tối ưu hóa, chúng ta thường làm việc với việc tối thiểu hóa hàm mất mát (loss function). Do đó, ta định nghĩa hàm mất mát $J(\theta)$ là âm của Log-Likelihood trung bình (Negative Log-Likelihood - NLL):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}; \theta)$$

Đây chính là nguồn gốc của hàm mất mát Cross-Entropy trong ngữ cảnh học máy.

3.2 Mối Liên Hệ Với Lý Thuyết Thông Tin (Information Theory)

Cross-Entropy (Entropy chéo) giữa hai phân phối xác suất p (thực tế) và q (dự đoán) được định nghĩa là:

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

Trong bài toán phân loại có giám sát:

- $p(x)$ là phân phối của nhãn thực tế y. Vì y là one-hot vector, $p(x) = 1$ tại lớp đúng và 0 tại các lớp khác.
- $q(x)$ là phân phối dự đoán $\hat{y}$ từ mô hình Softmax.

Do đó, Entropy chéo cho một mẫu dữ liệu đơn lẻ trở thành:

$$H(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

Vì y là one-hot, chỉ có thành phần $y_{target} = 1$ khác không, công thức rút gọn thành:

Điều này hoàn toàn trùng khớp với công thức NLL thu được từ MLE. Việc tối thiểu hóa Cross-Entropy chính là làm cho phân phối dự đoán $\hat{y}$ càng giống phân phối thực tế y càng tốt (tức là giảm thiểu khoảng cách Kullback-Leibler $D_{KL}(y||\hat{y})$)

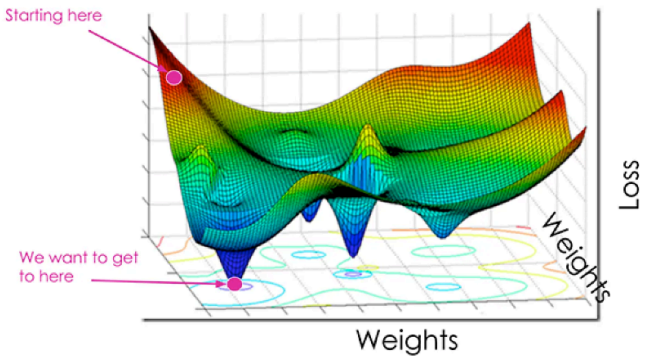
$$H(y, \hat{y}) = -\log(\hat{y}_{target})$$

3.3 Tại Sao Không Dùng Mean Squared Error (MSE)?

Một câu hỏi thường gặp (và được đề cập trong 7) là tại sao không dùng MSE ($L = \frac{1}{2}||\hat{y} - y||^2$) cho bài toán phân loại. Có hai lý do chính:

1. **Vấn đề Lồi** (Convexity): Khi kết hợp MSE với hàm kích hoạt phi tuyến như Softmax, hàm mất mát tạo thành bề mặt lỗi không lồi (non-convex), chứa nhiều cực tiểu địa phương (local minima). Điều này làm cho thuật toán Gradient Descent dễ bị mắc kẹt và không tìm được nghiệm tối ưu toàn cục. Ngược lại, Cross-Entropy kết hợp với Softmax là một hàm lồi (convex function) đối với logits z, đảm bảo tính hội tụ tốt hơn.
2. **Triệt Tiêu Gradient** (Vanishing Gradient): Khi dự đoán của mô hình sai lệch lớn so với thực tế (ví dụ: nhãn là 1 nhưng dự đoán gần 0), đạo hàm của MSE chứa thành phần $\sigma'(z)$ sẽ tiến về 0 rất nhanh (saturation), làm cho quá trình học bị đình trệ. Trong khi đó, Cross-Entropy có đặc tính tự điều chỉnh gradient: sai số càng lớn, gradient càng mạnh, giúp mô hình thoát khỏi trạng thái sai lầm nhanh chóng.

4 Đạo Hàm và Tối Ưu Hóa



Phần này sẽ trình bày chi tiết các bước tính đạo hàm (Backpropagation) cho Hồi quy Softmax. Đây là nội dung cốt lõi và cần được diễn giải tường tận để hiểu cơ chế cập nhật trọng số.

Ta cần tính gradient của hàm mất mát L đối với logit z_i, ký hiệu là $\frac{\partial L}{\partial z_i}$. Áp dụng quy tắc chuỗi (Chain Rule):

$$\frac{\partial L}{\partial z_i} = \sum_{j=1}^{K} \frac{\partial L}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial z_i}$$

Lưu ý rằng L phụ thuộc vào toàn bộ vector $\hat{y}$, và mỗi phần tử $\hat{y}_j$ lại phụ thuộc vào toàn bộ vector z (do mẫu số của Softmax chứa tổng các e^{z_k}). Do đó ta phải lấy tổng theo j.

Bước 1: Đạo Hàm Của Loss Theo $\hat{y}$ Với $L = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$, ta có:

$$\frac{\partial L}{\partial \hat{y}_j} = -\frac{y_j}{\hat{y}_j}$$

Bước 2: Đạo Hàm Của Softmax (Ma Trận Jacobian) Ta cần tính $\frac{\partial \hat{y}_j}{\partial z_i}$. Hàm Softmax: $\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$. Đặt $\Sigma = \sum_{k=1}^{K} e^{z_k}$.

- Trường hợp 1: $i = j$ Sử dụng quy tắc thương (quotient rule):

$$\frac{\partial \hat{y}_i}{\partial z_i} = \frac{(e^{z_i})' \cdot \Sigma - e^{z_i} \cdot (\Sigma)'}{\Sigma^2}$$

Vì đạo hàm của $\sum e^{z_k}$ theo z_i chỉ có số hạng e^{z_i} là khác 0, nên $(\Sigma)' = e^{z_i}$.

$$\frac{\partial \hat{y}_i}{\partial z_i} = \frac{e^{z_i}\Sigma - e^{z_i}e^{z_i}}{\Sigma^2} = \frac{e^{z_i}}{\Sigma}\left(1 - \frac{e^{z_i}}{\Sigma}\right) = \hat{y}_i(1 - \hat{y}_i)$$

- Trường hợp 2: $i \neq j$

$$\frac{\partial \hat{y}_j}{\partial z_i} = \frac{(e^{z_j})' \cdot \Sigma - e^{z_j} \cdot (\Sigma)'}{\Sigma^2}$$

Vì đạo hàm của e^{z_j} theo z_i là 0 (với $i \neq j$), ta có:

$$\frac{\partial \hat{y}_j}{\partial z_i} = \frac{0 - e^{z_j}e^{z_i}}{\Sigma^2} = -\frac{e^{z_j}}{\Sigma} \cdot \frac{e^{z_i}}{\Sigma} = -\hat{y}_j\hat{y}_i$$

- Tổng hợp:Sử dụng ký hiệu Kronecker delta ($\delta_{ij} = 1$ nếu $i = j$, ngược lại bằng 0):

$$\frac{\partial \hat{y}_j}{\partial z_i} = \hat{y}_j(\delta_{ij} - \hat{y}_i)$$

Đây chính là các phần tử của ma trận Jacobian của hàm Softmax. Ma trận này đối xứng và phụ thuộc vào chính giá trị đầu ra.

Bước 3: Kết Hợp (The Beautiful Cancellation) Thay thế kết quả từ **Bước 1** và **Bước 2** vào công thức chuỗi ban đầu:

$$\frac{\partial L}{\partial z_i} = \sum_{j=1}^{K} \left(-\frac{y_j}{\hat{y}j} \right) \cdot \hat{y}j(\delta ij - \hat{y}i)$$

$$= -\sum j = 1^K y_j(\delta ij - \hat{y}i)$$

$$= -\left(\sum j = 1^K y_j \delta_{ij} - \sum_{j=1}^{K} y_j \hat{y}_i \right)$$

Phân tích hai số hạng trong ngoặc:

1. $\sum_{j=1}^{K} y_j \delta_{ij}$: Chỉ khi $j = i$ thì $\delta_{ij} = 1$, các số hạng khác bằng 0. Vậy tổng này bằng y_i.
2. $\sum_{j=1}^{K} y_j \hat{y}_i = \hat{y}_i \sum_{j=1}^{K} y_j$: Vì y là vector one-hot xác suất, $\sum y_j = 1$. Vậy tổng này bằng $\hat{y}_i$.

Kết quả cuối cùng:

$$\frac{\partial L}{\partial z_i} = -(y_i - \hat{y}_i) = \hat{y}_i - y_i$$

Dưới dạng vector hóa:

$$\nabla_z L = \hat{y} - y$$

Đây là một kết quả toán học quan trọng trong học sâu: Gradient của hàm mất mát Cross-Entropy qua tầng Softmax chính là hiệu số giữa dự đoán và thực tế (prediction error). Điều này giải thích tại sao sự kết hợp Softmax + Cross-Entropy lại hiệu quả: sai số càng lớn, gradient càng lớn, tốc độ học càng nhanh, tránh được hiện tượng bão hòa gradient mà MSE gặp phải.

Cập Nhật Trọng Số (Weight Update) Sau khi có gradient theo logits $\delta = \hat{y} - y$, ta lan truyền ngược để tính gradient cho trọng số W và bias b:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial W} = x(\hat{y} - y)^T$$

$$\frac{\partial L}{\partial b} = \hat{y} - y$$

Quy tắc cập nhật Gradient Descent (với tốc độ học η):

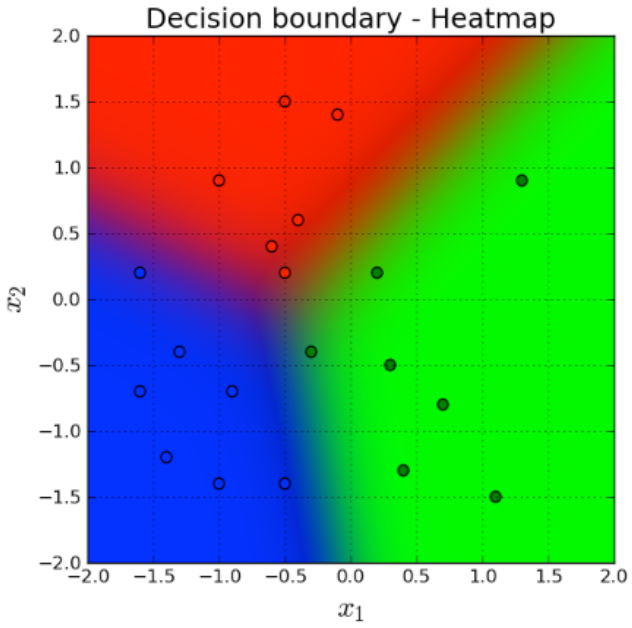
$$W \leftarrow W - \eta \cdot x(\hat{y} - y)^T$$

$$b \leftarrow b - \eta \cdot (\hat{y} - y)$$

5 Phân Tích Hình Học: Ranh Giới Quyết Định

5.1 Tính Tuyến Tính Của Ranh Giới (Linear Decision Boundaries)

Mặc dù hàm Softmax là phi tuyến, Hồi quy Softmax (trong dạng cơ bản không có lớp ẩn) là một bộ phân loại tuyến tính (Linear Classifier). Điều này có nghĩa là các đường ranh giới phân chia các lớp trong không gian đặc trưng là các đường thẳng (trong 2D) hoặc siêu phẳng (trong không gian chiều cao).



Chứng minh:

Ranh giới quyết định giữa hai lớp bất kỳ i và j là tập hợp các điểm x mà tại đó mô hình lưỡng lự giữa hai lớp, tức là xác suất dự đoán bằng nhau:

$$P(y = i|x) = P(y = j|x)$$

$$\frac{e^{w_i^T x + b_i}}{\sum_k e^{z_k}} = \frac{e^{w_j^T x + b_j}}{\sum_k e^{z_k}}$$

Khử mẫu số chung và lấy logarit tự nhiên hai vế:

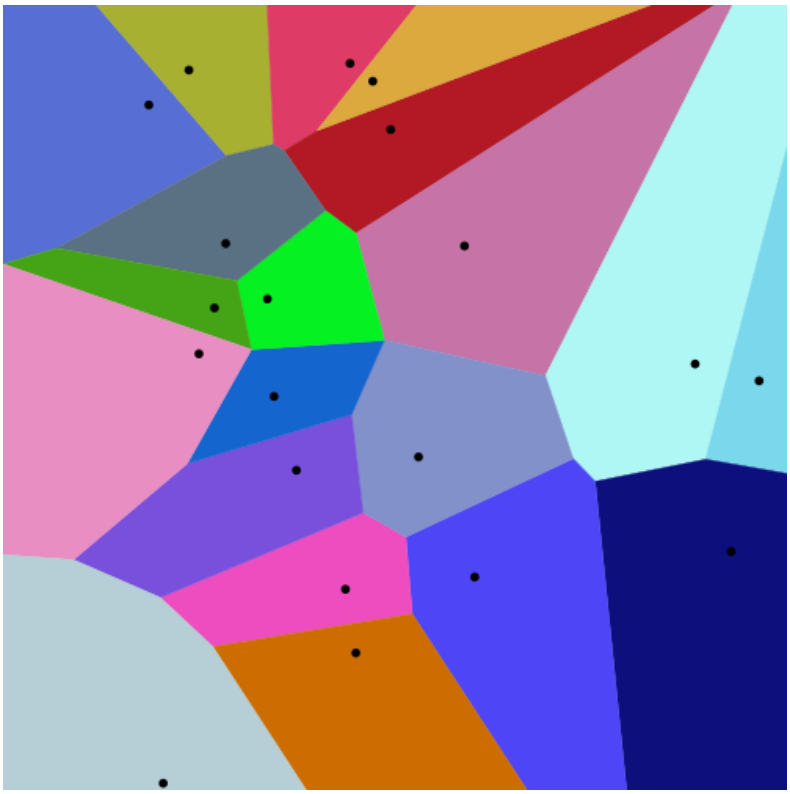
$$w_i^T x + b_i = w_j^T x + b_j$$

$$(w_i - w_j)^T x + (b_i - b_j) = 0$$

Phương trình trên có dạng $w'^T x + b' = 0$, đây chính xác là phương trình của một siêu phẳng (hyperplane). Điều này khẳng định rằng Hồi quy Softmax chia không gian đặc trưng thành các vùng tuyến tính.

5.2 Sơ Đồ Voronoi Tổng Quát

Các vùng quyết định của Hồi quy Softmax tạo thành một sơ đồ Voronoi tổng quát hóa. Mỗi vùng lồi (convex polytope) đại diện cho một lớp, nơi điểm số (logit) của lớp đó cao hơn tất cả các lớp còn lại. Điều này có ý nghĩa quan trọng: Hồi quy Softmax chỉ có thể giải quyết các bài toán mà dữ liệu phân tách tuyến tính (linearly separable). Đối với dữ liệu phức tạp (ví dụ: bài toán XOR, dữ liệu dạng xoắn ốc), ta bắt buộc phải sử dụng Feature Engineering hoặc Mạng Nơ-ron nhiều lớp (Deep Neural Networks) để biến đổi không gian đặc trưng trước khi đưa vào lớp Softmax cuối cùng.



6 Triển Khai Code

Việc triển khai Hồi quy Softmax đòi hỏi sự chú ý đặc biệt đến hiệu năng (vectorization) và độ ổn định (stability). Dưới đây là sự so sánh và hướng dẫn triển khai trên hai nền tảng phổ biến: NumPy (cho hiểu biết nền tảng) và PyTorch (cho ứng dụng thực tế).

6.1 Triển Khai Bằng NumPy (Vectorization Stability)

Dựa trên những gì đã trình bày và việc triển khai cần xử lý trên ma trận (batch processing) để tận dụng sức mạnh của BLAS/LAPACK.

Bảng 2: So sánh Naive Softmax và Stable Softmax

Đặc điểm	**Naive Softmax**	**Stable Softmax**
Công thức	np.exp(z) / np.sum(np.exp(z))	z -= np.max(z); np.exp(z) / np.sum(...)
Ưu điểm	Đơn giản, đúng định nghĩa toán học	Hoạt động ổn định với mọi giá trị đầu vào
Nhược điểm	Dễ bị inf/nan khi $z > 709$ (trong chuẩn float64)	Tốn thêm chi phí tìm max, nhưng an toàn tuyệt đối

```python
import numpy as np

class SoftmaxRegression:
    def __init__(self, n_features, n_classes, lr=0.01):
        # ởKhi ạto ọtrng ố ỏnh ẫngu nhiên, bias ằbng 0
        self.W = np.random.randn(n_features, n_classes) * 0.01
        self.b = np.zeros((1, n_classes))
        self.lr = lr

    def forward(self, X):
        """
        Tính toán xuôi: z = XW + b -> softmax(z)
        Input: X (m, d)
        Output: y_hat (m, K)
        """
        z = np.dot(X, self.W) + self.b

        # Stable Softmax: ừtr đi max ủca ỗmi hàng
        z_max = np.max(z, axis=1, keepdims=True)
        exp_z = np.exp(z - z_max)
        y_hat = exp_z / np.sum(exp_z, axis=1, keepdims=True)
        return y_hat
```

```
    def compute_loss(self, y_hat, y_indices):
        """
        Tính Cross-Entropy Loss
        Input: y_hat (m, K), y_indices (m,) ứcha nhãn ớlp (0..K-1)
        """
        m = y_indices.shape[0] # uLu ý: ửsa code ốgc (shape ảtr ềv tuple)
        # ấLy xác ấsut ựd đoán ủca ớlp đúng: y_hat[range(m), y_indices]
        # Thêm epsilon để tránh log(0)
        correct_logprobs = -np.log(y_hat[range(m), y_indices] + 1e-15)
        loss = np.sum(correct_logprobs) / m
        return loss

    def backward(self, X, y_hat, y_indices):
        """
        Tính gradient và ậcp ậnht tham ốs
        """
        m = X.shape[0]

        # 1. Tính gradient ủca Loss theo z: dz = y_hat - y_onehot
        dz = y_hat.copy()
        # ừTr 1 ạti ịv trí ớlp đúng (ươtng đương y_onehot)
        dz[range(m), y_indices] -= 1
        dz /= m  # Chia cho m để ấly trung bình gradient

        # 2. Gradient theo W và b
        dW = np.dot(X.T, dz)
        db = np.sum(dz, axis=0, keepdims=True)

        # 3. ậCp ậnht (Gradient Descent)
        self.W -= self.lr * dW
        self.b -= self.lr * db
```

Mã nguồn trên tích hợp các kỹ thuật quan trọng: trừ max để ổn định hàm mũ, dùng indexing mảng (y_hat[range(m), y_indices]) để tránh tạo ma trận one-hot lớn tốn bộ nhớ (sparse implementation), và thêm epsilon vào logarit.

6.2 Triển Khai Bằng PyTorch: CrossEntropyLoss vs NLLLoss

Trong hệ sinh thái PyTorch, có một sự phân chia tinh tế nhưng cực kỳ quan trọng về các hàm mất mát liên quan đến Softmax, thường gây nhầm lẫn cho người mới.

Bảng 3: Các hàm Loss phổ biến trong PyTorch

Hàm PyTorch	**Đầu Vào Mong Đợi**	**Cơ Chế Bên Trong**	**Khi Nào Dùng?**
nn.CrossEntropyLoss	**Logits** (chưa qua Softmax)	LogSoftmax + NLLLoss	**Khuyên dùng** cho đa số bài toán phân loại đa lớp. Ổn định số học nhất.
nn.NLLLoss	**Log-Probabilities** (đã qua LogSoftmax)	Tính tổng âm log xác suất theo nhãn	Dùng khi mô hình có lớp cuối là nn.LogSoftmax.
nn.BCELoss	Xác suất (đã qua Sigmoid)	Binary Cross Entropy	Chỉ dùng cho phân loại nhị phân (Binary Classification).

Tại sao `nn.CrossEntropyLoss` lại tốt hơn?

Hàm này thực hiện tính toán `LogSoftmax` trong một bước duy nhất bằng kỹ thuật Log-Sum-Exp trick đã phân tích ở Mục 3.3. Nó tránh việc tính e^z (có thể rất lớn) rồi mới tính log, mà thay vào đó tính toán trực tiếp trên miền logarit, giữ cho các giá trị số học luôn nằm trong phạm vi an toàn. Nếu người dùng tự cài đặt `Softmax` → `Log` → `NLLLoss`, nguy cơ gặp `nan` do underflow ở bước `Log` là rất cao.

Ví dụ Code PyTorch chuẩn:

```python
import torch
import torch.nn as nn
import torch.optim as optim

# Gia su du lieu
X_train = torch.randn(100, 4)  # 100 mau, 4 dac trung
y_train = torch.randint(0, 3, (100,))  # 3 lop (0, 1, 2)

# Mo hinh: Chi can lop Linear, KHONG can Softmax o cuoi
model = nn.Linear(4, 3)

# Ham loss tu dong xu ly LogSoftmax
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)

# Vong lap huan luyen
for epoch in range(100):
    optimizer.zero_grad()

    # Forward: Ket qua la Logits, khong phai xac suat
    logits = model(X_train)

    # Loss: CrossEntropyLoss nhan logits va class indices (khong phai one-hot)
    loss = criterion(logits, y_train)

    loss.backward()
    optimizer.step()

    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")

# Khi du doan (Inference), moi can dung Softmax de lay xac suat
with torch.no_grad():
    test_logits = model(X_train[:5])
    probs = torch.softmax(test_logits, dim=1)
    predictions = torch.argmax(probs, dim=1)
```

Kết luận

Blog này đã trình bày một cái nhìn toàn diện và sâu sắc về Hồi quy Softmax, đi từ động lực trực giác đến các chứng minh toán học chặt chẽ và kỹ thuật triển khai tối ưu.

Các điểm cốt lõi cần ghi nhớ:

- **Sự Tổng Quát Hóa**: Softmax là sự mở rộng tự nhiên của Logistic Sigmoid cho đa lớp. Nó chuyển đổi các điểm số tự do (logits) thành phân phối xác suất hợp lệ thông qua hàm mũ và chuẩn hóa.
- **Sức Mạnh Của Cross-Entropy**: Hàm mất mát này, bắt nguồn từ nguyên lý MLE và Lý thuyết thông tin, cung cấp độ dốc gradient mạnh mẽ ($\hat{y} - y$) ngay cả khi sai số lớn, khắc phục nhược điểm của MSE trong phân loại.
- **Ổn Định Số Học Là Chìa Khóa**: Trong thực tế kỹ thuật, việc cài đặt "Naive Softmax" là nguy hiểm. Kỹ thuật "Subtract Max" và "Log-Sum-Exp" là bắt buộc để đảm bảo mô hình hoạt động bền vững trên phần cứng máy tính.
- **Tuyến Tính Nhưng Mạnh Mẽ**: Mặc dù ranh giới quyết định là tuyến tính, Hồi quy Softmax là thành phần không thể thiếu – là "cửa ngõ" cuối cùng – của các mô hình học sâu phức tạp nhất hiện nay (như Transformer, CNN), nơi các lớp ẩn chịu trách nhiệm biến đổi dữ liệu phi tuyến thành không gian phân tách tuyến tính cho Softmax xử lý.

Thông qua việc nắm vững cơ chế của Softmax và Cross-Entropy, người thực hành không chỉ hiểu rõ "hộp đen" của các thư viện như PyTorch mà còn có khả năng chẩn đoán lỗi, tối ưu hóa hiệu năng và thiết kế các kiến trúc mô hình mới một cách hiệu quả.