AI VIETNAM – COURSE 2023

Softmax Regression

Ngày 28 tháng 5 năm 2023

1 Giới thiệu

Softmax Regression là một thuật toán Machine learning cơ bản thuộc nhánh Supervised Learning. Tương tự như Logistic Regression, Softmax Regression cũng được dùng để giải quyết các bài toán phân lớp (classification). Tuy nhiên, trong khi về cơ bản Logistic Regression chỉ có thể áp dụng để giải quyết các bài toán phân lớp nhị phân (có 2 class), Softmax Regression lại có thể giải quyết các bài toán có số lượng class lớn hơn 2.

2 Các cải thiện của Softmax regression

- **Xử lý đa lớp**:
  Hàm logistic có thể sử dụng được cho phân loại đa lớp khi mình áp dụng One vs Rest (cho một class bất kì làm một loại và cho tất cả các class còn lại về một loại)
  Hàm softmax có thể tính toán xác suất cho nhiều lớp cùng lúc nên hàm softmax sẽ có liên kết giữa các output chặt chẽ hơn One vs Rest

- **Xác suất đầu ra chuẩn hóa**:
  Hàm softmax đảm bảo rằng tổng các xác suất dự đoán cho tất cả các lớp là 1. Điều này rất hữu ích trong việc tạo ra một phân phối xác suất chính xác trên các lớp.

- **Phép đo lường mất mát**:
  Khi sử dụng softmax, chúng ta thường sử dụng hàm cross-entropy để đo lường sự khác biệt giữa phân phối xác suất ước tính và nhãn thực tế. Cross-entropy thường là một phép đo lường thông tin hiệu quả và thường được sử dụng như hàm mất mát trong quá trình huấn luyện mô hình. Trong khi đó, hàm logistic dùng binary cross-entropy là một trường hợp đặc biệt của cross-entropy

- **Tích hợp trong mô hình machine learning**:
  Softmax được sử dụng rộng rãi trong các mô hình học máy và các mạng nơ-ron nhân tạo để xử lý bài toán phân loại đa lớp. Việc tích hợp softmax trong mô hình học máy tạo điều kiện thuận lợi cho việc huấn luyện và đánh giá mô hình.

Thuật toán Softmax cải thiện khả năng xử lý từ hàm logistic để tính toán xác suất dự đoán trong bài toán phân loại đa lớp và là một phần quan trọng trong quá trình huấn luyện và đánh giá mô hình phân loại đa lớp. Ngoài ra khi quy về 2 class, hàm softmax sẽ giống như hàm sigmoid.

3 Softmax Regression

Định nghĩa 3.1 *Hồi quy Softmax là một dạng hồi quy logistic chuẩn hóa một giá trị đầu vào thành một vector các giá trị tuân theo phân phối xác suất có tổng bằng 1. Các giá trị đầu ra nằm trong khoảng*

[0,1], có thể tránh phân lớp nhị phân (binary classification) và có thể sử dụng nhiều lớp, chiều trong mô hình neural network. Đây là lý do tại sao Softmax đôi khi được gọi là hồi quy logistic đa thức.

Công thức 3.1

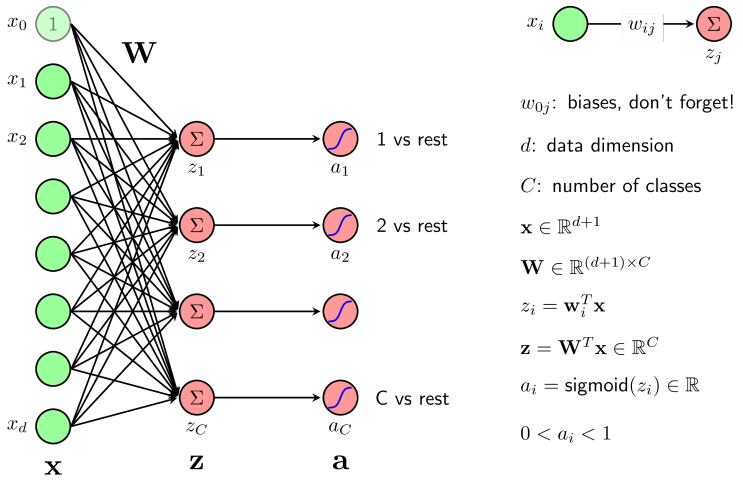
$$P(c = j|\theta^{(i)}) = \frac{e^{\theta^{(i)}}}{\sum_{j=0}^{C} e^{\theta_c^{(i)}}} \tag{1}$$

Với $\theta = w_0x_0 + w_1x_1 + \cdots + w_Cx_C = \sum_{i=0}^{C} = w^Tx$

Giải thích công thức

- Mục tiêu là một tập các feature x (biểu diễn dạng vector hàng, gọi là ma trận đặc trưng) với các trọng số w (dạng vector cột, gọi là ma trận đặc trưng) thuộc class j
- Tham số đầu vào θ là ma trận được tạo từ one-hot encoding hoặc integer encoding (lable encoding).
- Mẫu số là 1 normalization term, bảo đảm giá trị output nằm giữa 0 và 1. Lí do vì sao không dùng chuẩn hoá thông thường (như dùng logit)
  - Chuẩn hoá như ở công thức Softmax sẽ phản ứng khác nhau với độ lớn của sự biến đổi, còn chuẩn hoá tiêu chuẩn sẽ không phân biệt kết quả với tỉ lệ giống nhau.
  - Dùng logit sẽ gây ra vấn đề nếu giá trị trả về là âm. Còn dùng mũ thì giá trị luôn dương.
- Trong công thức, ta tính hàm mũ của tham số đầu vào và tổng các tham số hàm mũ của tất cả các giá trị. Đầu ra của công thức là tỉ lệ của hai giá trị này.

Trong hồi quy logistic nhị phân, chúng ta giả định rằng các nhãn là nhị phân nhưng các bài toán classification thực tế thường có rất nhiều classes (multi-class), các binary classifiers mặc dù có thể áp dụng cho các bài toán multi-class, chúng vẫn có những hạn chế nhất định. Với binary classifiers, kỹ thuật được sử dụng nhiều nhất one-vs-rest thì output layer có thể phân tách thành hai sublayer như hình dưới đây:



Hình 1: Multi-class classification với one-vs-rest

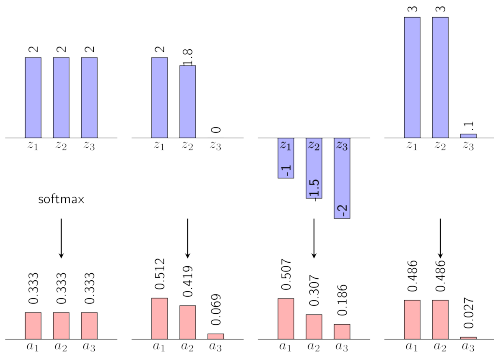
Giả sử số classes là C. Với one-vs-rest chúng ta cần xây dựng C Logistic Regression khác nhau. Các đầu ra dự đoán được tính theo hàm sigmoid:

$$a_i = sigmoid(z_i) = sigmoid(w_i^T x) \tag{2}$$

Trong trường hợp này, các phần tử a_i được suy ra trực tiếp từ z_i. Vì vậy, mối quan hệ giữa các a_i không chặt chẽ (tổng của chúng có thể nhỏ hơn hoặc lớn hơn 1). Để thỏa mãn điều kiện là các a_i dương và tổng của chúng bằng 1. Ngoài ra, còn điều kiện khi giá trị z_i càng lớn thì xác suất dữ liệu rơi vào class i càng cao. Hàm thỏa mãn các điều kiện trên là hàm e^{z_i}, để đảm bảo rằng tổng các a_i bằng 1 khi:

$$a_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{3}$$

Hàm 3 gọi là hàm softmax function, với i có giá trị từ 1 đến C ($j \neq i$). Khi giá trị z_i rất nhỏ hoặc rất lớn khi so sánh với các z_j thì giá trị a_i có thể rất gần 0 hoặc 1, nhưng không có giá trị tuyệt đối bằng 0 hoặc tuyệt đối bằng 1.



Hình 2: Input và Output khi dùng softmax function

4 Loss function

4.1 Log-Likelihood

Hàm Softmax sẽ trả về một giá trị dự đoán ($\hat{y}$) mà có thể hiểu là xác suất có điều kiện của mỗi lớp có đầu vào x được biểu diễn bằng cách sử dụng một vectơ one-hot encoding có thể so sánh các ước tính với thực tế bằng cách kiểm tra xem các lớp thực tế có thể xảy ra như thế nào theo mô hình.

$$P(Y|X) = \prod_{i=1}^{n} P(y^i|x^i) \tag{4}$$

Giả định rằng mỗi nhãn độc lập với phân phối của nó, vì việc tối đa hóa các số hạng là khó nên lấy logarit âm để có được bài toán tương đương về cực tiểu hóa log-likelihood âm.

$$-logP(Y|X) = \sum_{i=1}^{n} -logP(y^i|x^i) = \sum_{i=1}^{n} l(y^i, \hat{y}), \tag{5}$$

Log-Likelihood (còn được gọi là Cross-Entropy Loss). Nó được sử dụng để đánh giá độ chính xác của mô hình softmax regression và tối ưu hóa các tham số của mô hình cho nhãn y giá trị dự đoán x trên q lớp thi hàm mất mát.

$$l(y, \hat{y}) = -\sum_{i=1}^{q} y_j log\hat{y}_i \tag{6}$$

4.2 Softmax và Cross-Entropy Loss

$$l(y, \hat{y}) = -\sum_{j=1}^{q} y_j log\frac{exp(o_j)}{\sum_{k=1}^{q} exp(o_k)} \tag{7}$$

$$= \sum_{j=1}^{q} y_j log\sum_{k=1}^{q} exp(o_k) - \sum_{j=1}^{q} y_j o_j \tag{8}$$

$$= log\sum_{k=1}^{q} exp(o_k) - \sum_{j=1}^{q} y_j o_j \tag{9}$$

Xem xét đạo hàm với output o_j:

$$\partial_{o_j} l(y, \hat{y}) = \frac{exp(o_j)}{\sum_{k=1}^{q} exp(o_k)} - y_i \tag{10}$$

$$= softmax(o)_j - y_j \tag{11}$$

5 Overflow trong hàm Softmax

5.1 Định nghĩa về Floating-point arithmetic

Trong tin học, dấu phẩy động (Floating-point arithmetic) được dùng để chỉ một hệ thống biểu diễn mà trong đó sử dụng chữ số cơ số 10 (hay bit) để biểu diễn xấp xỉ 1 số thực, trong đó sử một số nguyên cố định được gọi là **phần định trị (significand)** và được chia tỉ lệ với **số mũ (exponent)** với một cơ số cố định,

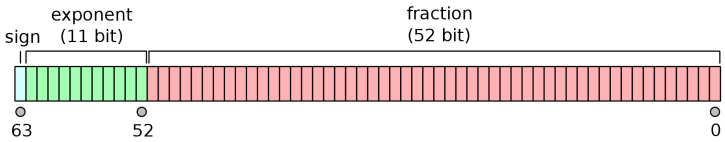
Ví dụ ta biểu diễn số 12.345 dưới dạng cơ số 10 với dấu phẩy động:

Có nhiều hệ thống dấu phẩy động khác nhau được dùng trong máy tính; tuy nhiên, trong khoảng 20 năm gần đây hầu hết các máy tính đều dùng cách biểu diễn tuân thủ theo quy chuẩn của IEEE 754. Với tiêu chuẩn của IEEE 754 cho binary64 ta có:

1. bit dấu: 1 bit
2. bit mũ: 11 bit
3. bit phân số: 53 bit (nhưng dùng 52 bit để lưu trữ)

Ta có thể biểu diễn với dạng công thức như sau:

$$(-1)^{sign}(1.b_{51}b_{50}\dots b_0)_2 \times 2^{e-1023}$$

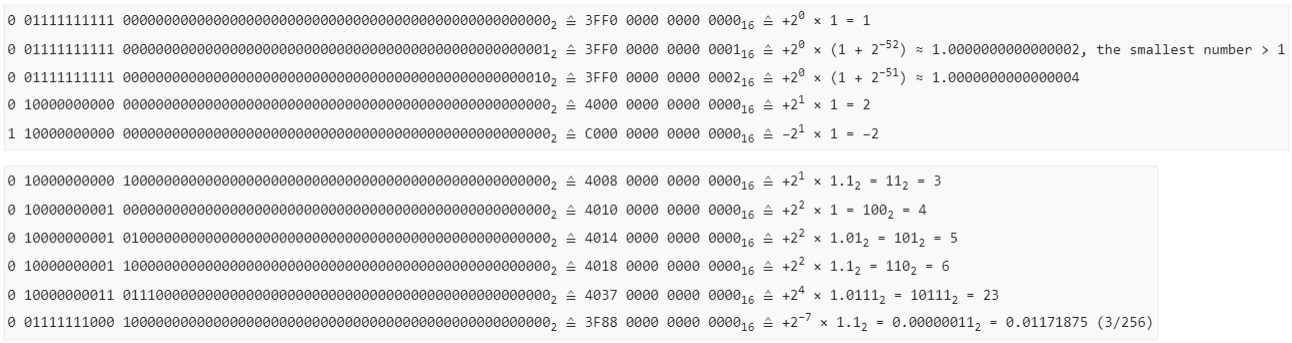


Hình 3: IEEE 754

or

$$(-1)^{sign}(1+\sum_{i=1}^{52} b_{52-1}2^{-i}) \times 2^{e-1023}$$

Một số ví dụ biểu diễn floating-poitn với IEEE 754:



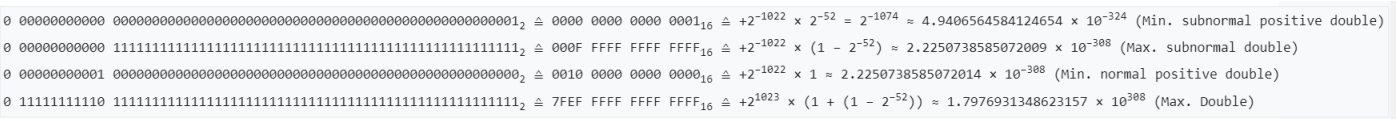
Hình 4: Ví dụ biểu diễn IEEE 754

5.2 Định nghĩa về Overflow và vì sao Floating-point arithmetic lại gây ra Overflow

Overflow là một lỗi số có hại thường gặp phải khi lập trình. Lỗi này xảy khi số quá lớn $-\infty$ hay ∞ làm cho kiểu dữ liệu của ngôn ngữ lập trình đang dùng không thể nào biểu diễn được nó thì sẽ dẫn đến lỗi Overflow, ví dụ: float trong python có thể biểu diễn tối đa là $1.7976931348623157 * 10^{308}$, thì khi ta cho gán 10^{1000} hay -10^{1000} thì kết quả sẽ trả về là inf hoặc -inf và ta không thể dùng kết quả này để tính toán Softmax được.

Như đã giới thiệu về dấu phẩy động ở trên phần lớn các ngôn ngữ lập trình đều dùng chuẩn IEEE 754 để biểu diễn giá trị của các số thực vậy nên giới hạn 1 số có thể lưu cũng phụ thuộc vào chuẩn này là binary64 nó tương đương với việc các biến có thể lưu được trong khoảng từ $2^{-1022} \approx 2 \times 10^{-308}$ đến $2^{1024} \approx 2 \times 10^{308}$.

Ta có thể biểu diễn nó theo dãy bit như sau:



Hình 5: Giới hạn của IEEE 754 với binary64

Vậy nên nếu ta lưu 1 số vượt quá giới hạn này thì ngôn ngữ lập trình không thể hiểu và biểu diễn 1 cách chính xác được dẫn đến việc tràn số. Vì hàm Softmax là hàm có mũ nên việc tràn số là việc hoàn toàn có thể xảy ra nếu số đầu vào lớn hơn 300.

5.3 Cách khắc phục Overflow trong Softmax

Với công thức cơ bản của hàm Softmax:

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)}, \forall_i = 1, 2, \ldots, C \tag{12}$$

Code mẫu với python cho công thức ở trên:

```
import numpy as np

def softmax(Z):
    e_Z = np.exp(Z)
    A = e_Z / e_Z.sum(axis= 0)
    return A
```

Khi z_i quá lớn có thể dẫn đến việc bị overflow, do ta sẽ sử dụng 1 công thức khác của hàm Softmax để khắc phục điểm yếu này:

$$\frac{\exp(z_i)}{\sum_{j=1}^{C} exp(z_j)} = \frac{\exp(-c)\exp(z_i)}{\sum_{j=1}^{C} \exp(-c)\exp(z_j)} = \frac{\exp(z_i - c)}{\sum_{j=1}^{C} \exp(z_j - c)} \tag{13}$$

với c là 1 hằng số bất kỳ. Và trong thức nghiệm chúng ta sẽ giá đủ lớn với $c = max_i z_i$, qua đó giúp khắc phục được sự tràn số của hàm Softmax Dưới đây là code mẫu với python cho công thức với khắc phục Overflow cho Softmax:

```
import numpy as np

def softmax_stable(Z):
    """
    Compute softmax values for each sets of scores in Z.
    each column of Z is a set of score.
    """
    e_Z = np.exp(Z - np.max(Z, axis = 0, keepdims = True))
    A = e_Z / e_Z.sum(axis = 0)
    return A
```