ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

NGÀNH: KHOA HỌC DỮ LIỆU

MÔN HỌC: MACHINE LEARNING

Xây dựng các mô hình học máy với bộ dữ liệu MNIST

Sinh viên:

Nguyễn Duy Anh – 20002028

Lê Thế Cường – 20002035

Mục lục

Lời nói đầu

Học máy là một lĩnh vực của trí tuệ nhân tạo và đã trở thành một trong những chủ đề hot nhất trong ngành công nghiệp, khoa học và kinh doanh trong những năm gần đây. Học máy cho phép chúng ta xây dựng các mô hình dự đoán từ dữ liệu, giúp giải quyết các vấn đề phức tạp trong nhiều lĩnh vực như tài chính, y tế, bán lẻ, sản xuất, năng lượng và nhiều lĩnh vực khác. Nó giúp chúng ta rút ra những thông tin quan trọng từ dữ liệu lớn, phân tích dữ liệu và dự đoán xu hướng tương lai.

Với sự phát triển của công nghệ, học máy có thể được áp dụng vào rất nhiều lĩnh vực khác nhau, từ các thiết bị thông minh đến tự động hóa và robot học. Việc ứng dụng học máy đang trở thành một trong những phương tiện quan trọng giúp các tổ chức tăng cường năng suất, cải thiện hiệu quả và tăng tính cạnh tranh. Chính vì vậy, ngay từ đầu thì việc nắm rõ và hiểu biết sâu sắc một số thuật toán học máy kinh điển là một việc hết sức quan trọng. Nhóm chúng em xin gửi lời cảm ơn tới thầy Cao Văn Chung, giảng viên trực tiếp giảng dạy bộ môn Học máy và đã tạo điều kiện cho chúng em thực hiện đề tài về việc xây dựng mô hình và khai phá bộ dữ liệu chữ viết tay chuẩn (MNIST) để chúng em có thêm những sự hiểu biết sâu đối với môn học này.

1 Giới thiệu đề tài

1.1 Tổng quan về đề tài và phương pháp giải quyết

Đề tài của nhóm chúng em là xây dựng một mô hình dự đoán chữ số viết tay trên bộ dữ liệu MNIST. Ngoài ra, nhóm chúng em cũng sẽ xây dựng những mô hình học không giám sát như việc giảm số chiều và phân cụm để tìm hiểu thêm về bộ dữ liệu này. Có thể nói, MNIST là một bộ dữ liệu này được sử dụng rộng rãi trong lĩnh vực học máy và thường được sử dụng như một bài toán thử nghiệm cho các mô hình phân loại ảnh.

Để giải quyết được bài toán này, chúng em sẽ tiến hành sử dụng hai thuật toán học máy phổ biến, đó là Multi-Logistic Regression (Softmax) và và mạng Neural nhân tạo (ANN) để huấn luyện các mô hình dự đoán. Sau đó, chúng em sẽ đánh giá hiệu suất của các mô hình dựa trên các thông số về độ chính xác (Accuracy) và ma trận lỗi (Confusion matrix), và tối ưu hóa các siêu tham số để cải thiện hiệu suất của mô hình.

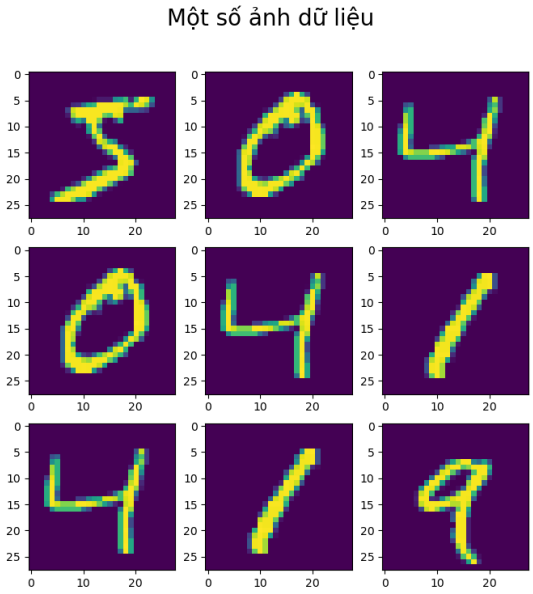
Mục tiêu của đề tài này là xây dựng một mô hình phân loại chính xác và hiệu quả dựa trên bộ dữ liệu MNIST. Kết quả của đề tài sẽ đóng góp vào việc cải thiện các ứng dụng của học máy trong việc phân loại ảnh và đưa ra những giải pháp hiệu quả cho các vấn đề thực tế khác.

1.2 Giới thiệu về bộ dữ liệu MNIST

MNIST – là viết tắt của Modified National Institute of Standards and Technology database. Về mặt lịch sử được tạo ra bởi ba nhà khoa học người Mỹ, Yann LeCun, Corinna Cortes và Christopher J.C. Burges, khi họ làm việc tại AT&T Bell Laboratories vào năm 1998. Ban đầu, bộ dữ liệu này được sử dụng để đào tạo mô hình phân loại chữ số viết tay cho hệ thống fax tự động của AT&T. Sau đó, bộ dữ liệu này được công bố công khai và trở thành một trong những bộ dữ liệu phổ biến nhất và được sử dụng rộng rãi trong lĩnh vực học máy. Bộ dữ liệu MNIST được sử dụng rộng rãi trong lĩnh vực học máy và nhất là trong việc đào tạo các mô hình học sâu. Nó cung cấp một cơ sở thử nghiệm tốt cho các mô hình phân loại ảnh và cho phép các nhà nghiên cứu so sánh hiệu suất của các thuật toán học máy khác nhau trên cùng một bộ dữ

liệu.

Về đặc điểm, bộ dữ liệu MNIST chứa hình ảnh đen trắng của các chữ số viết tay từ 0 đến 9, mỗi hình ảnh có kích thước là 28x28 pixel. Bộ dữ liệu gốc gồm 60,000 hình ảnh để huấn luyện và 10,000 hình ảnh để kiểm tra. Sau đây là một số hình ảnh về bộ dữ liệu.

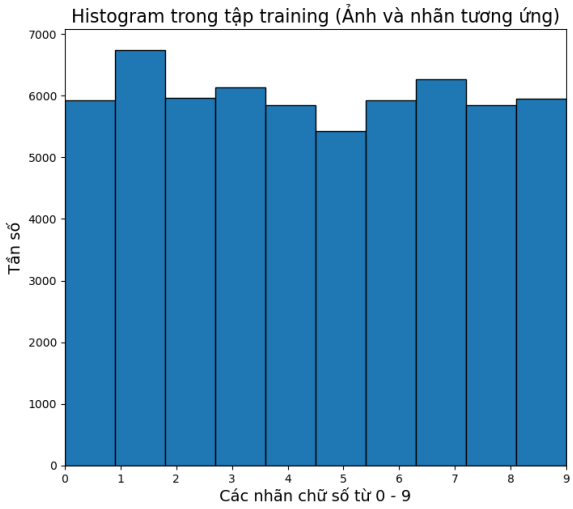


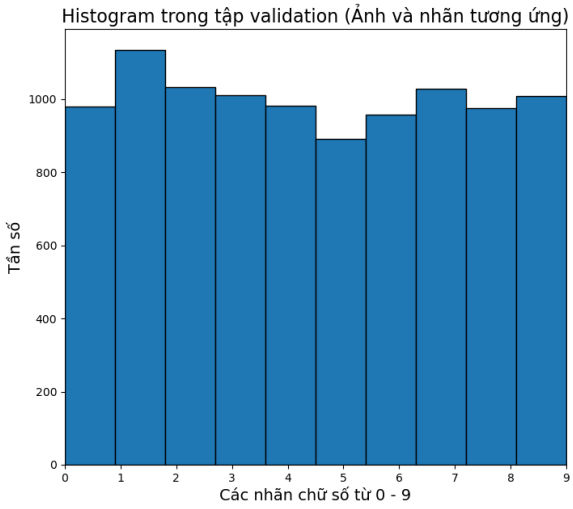
Về số lượng của các thành phần nhãn trong bộ dữ liệu, ta có các bảng thống kê sau:

Số (Nhãn)	Tần số (tập Training)	Tỉ lệ (tập Training)	Tần số (tập Validation)	Tỉ lệ (tập Validation)
0	5923	9.87%	980	9.80%
1	6742	11.24%	1135	11.35%
2	5958	9.93%	1032	10.32%
3	6131	10.22%	1010	10.10%
4	5842	9.74%	982	9.82%
5	5421	9.04%	892	8.92%
6	5918	9.86%	958	9.58%
7	6265	10.44%	1028	10.28%
8	5851	9.75%	974	9.74%
9	5949	9.92%	1009	10.09%

Từ bảng trên ta có thể vẽ ta lược đồ tần số (histogram) đối với hai bộ dữ liệu huấn luyện

(training) và kiểm thử (validation) để thấy rõ hơn sự phân bố đối với các nhãn trong hai bộ dữ liệu này.





Từ biểu đồ histgram cùng với tỷ lệ phần trăm số lượng của các nhãn trong hai bộ dữ liệu, cho thấy các bộ dữ liệu khá đồng đều về số lượng các nhãn, không có dữ liệu nhãn nào quá bị lệch so với các nhãn còn lại. Điều đó cho thấy, bộ dữ liệu MNIST của chúng ta đã được chuẩn hoá, có sự đồng đều trong phân bố các nhãn ở cả tập dữ liệu training và validation.

2 Giảm chiều và phân cụm cho bộ dữ liệu MNIST

2.1 Xây dựng mô hình giảm chiều bằng phương pháp PCA và hiển thị trực quan

Giảm chiều dữ liệu là một trong những kỹ thuật quan trọng của Machine Learning. Các feature trong các bài toán thực tế có số chiều lớn, tới vài nghìn. Ngoài ra, số điểm dữ liệu cũng thường rất lớn. Nếu thực hiện lưu trữ, thực hiện các mô hình học máy thì sẽ gặp khó khăn về cả việc lưu trữ và tốc độ tính toán.

Dimensionality Reduction, một cách đơn giản là ta đi tìm một ánh xạ, với đầu vào là điểm dữ liệu $x_i \in R^D$ với D rất lớn, tạo ra một điểm dữ liệu mới $z \in R^K (K < D)$

Một thuật toán cơ bản nhất trong Dimensionality Reduction dựa trên một mô hình tuyến tính. Phương pháp này có tên là **Principal Compoment Analysis (PCA)**, tức Phân tích thành phần chính.

Phương pháp này dựa trên quan sát rằng dữ liệu thường không phân bố ngẫu nhiên trong không gian mà thường phân bố gần các đường hoặc các mặt đặc biệt nào đó.

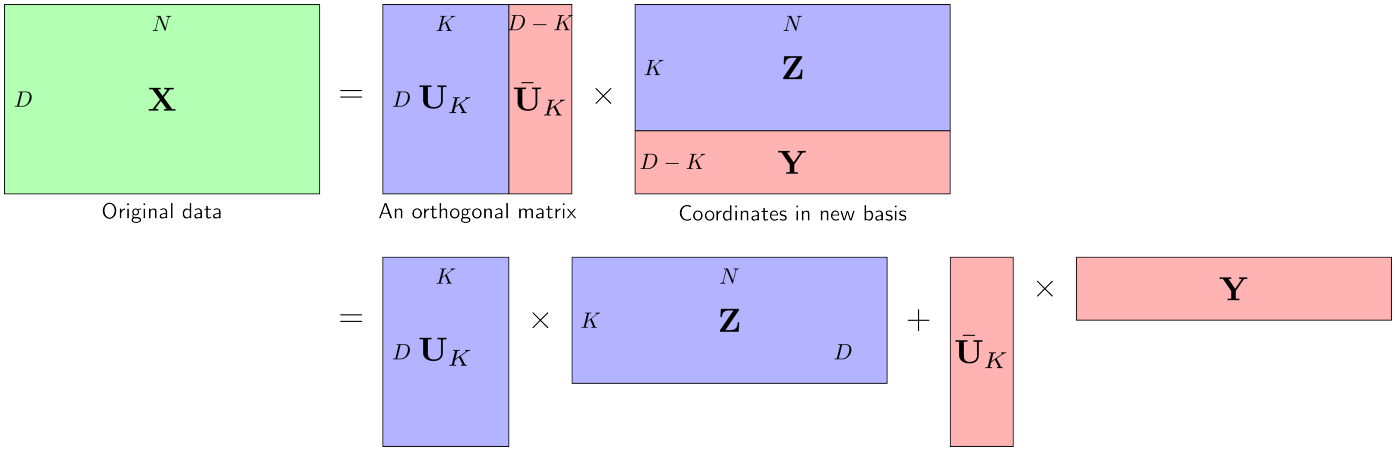
PCA chính là việc đi tìm một hệ cơ sở mới sao cho thông tin của dữ liệu chủ yếu tập trung ở một vài tọa độ, phần còn lại chỉ mang lượng nhỏ thông tin có thể chấp nhận bỏ đi được. Và để đơn giản cho trong tính toán. PCA sẽ đi tìm một hệ trực chuẩn để làm cơ sở mới.

2.1.1 Tóm tắt cơ sở lý thuyết và thuật toán giảm chiều PCA

Như đã đề cập ở trên. PCA sẽ đi tìm một hệ cơ sở trực chuẩn để làm cơ sở mới. Vậy, ta giả sử hệ cơ sở trực chuẩn mới là U và chúng ta muốn giữ lại K tọa độ trong hệ cơ sở mới. Không mất tổng quát, giả sử đó là K thành phần đầu tiên.

Hình trên cho thấy hệ cơ sở mới $U = [U_K, \overline{U}_K]$. U là một hệ trực chuẩn U_K với ma trận được tạo bởi K cột đầu tiên của U. Với cơ sở mới này, dữ liệu có thể được viết thành.

$$X = U_K Z + \bar{U}_K Y \quad (1)$$



Ta cũng có thể viết dưới dạng sau

$$X = \begin{bmatrix} U_K & \bar{U}_k \end{bmatrix} \begin{bmatrix} Z \\ Y \end{bmatrix}$$

$$-> \begin{matrix} U_k^T X = Z \\ \bar{U}_K^T X = Y \end{matrix} \qquad (2)$$

Chúng ta cố gắng giữ lại nhiều thông tin nhất có thể ở phần $U_K Z$ và lược bỏ phần $\bar{U_K} Y$. Để phần lược bỏ đi không phụ thuộc vào từng điểm dữ liệu. Ta sẽ xấp xỉ Y bởi một ma trận có toàn bộ các cột là như nhau. Gọi mỗi cột đó là b và có thể coi nó là bias. Khi đó, ta sẽ xấp xỉ:

$$Y \approx b1^T$$

Trong đó $1^T \in R^{1 \times N}$ là các hàng có toàn bộ phần tử bằng 1. Giả sử tìm được U, ta cần tìm b thỏa mãn:

$$b = argmin \|Y - b1^T\|_F^2 = argmin \|\bar{U}_K^T X - b1^T\|_F^2$$

Giải phương trình trên theo đạo hàm của b với hàm mục tiêu:

$$-> b = \bar{U}_K^T \bar{x}$$

Việc tính toán sẽ thuận tiện nếu vecto kỳ vọng $\bar{x} = 0$. Việc này có thể thực hiện ngay từ đầu (Trừ mỗi vector cho kỳ vọng của toàn bộ dữ liệu), bước đầu tiên của PCA.

Với giá trị b tìm được, dữ liệu ban đầu được xấp xỉ với:

$$X \approx \widetilde{X} = U_k Z + \bar{U}_k \bar{U}_K^T \bar{x} 1^T \qquad (3)$$

Kết hợp (1) (2) (3) ta định nghĩa hàm mất mát như sau:

$$J = \frac{1}{N} \|X - \widetilde{X}\|_F^2 = \frac{1}{N} \|\bar{U}_K \bar{U}_K^T X - \bar{U}_K \bar{U}_K^T \bar{x} 1^T\|_F^2 \quad (4)$$

Hàm (4) có thể được viết lại thành:

$$J = \frac{1}{N} \sum_{i=K+1}^{D} \|\hat{X}^T u_i\|_2^2$$

$$= \frac{1}{N} \sum_{i=K+1}^{D} u_i^T \hat{X} \hat{X}^T u_i$$

$$= \sum_{i=K+1}^{D} u_i^T \hat{X} \hat{X}^T u_i$$

$$= \sum_{i=K+1}^{D} u_i^T S u_i \quad (5)$$

Với $\hat{X} = X - x1^T$ là dữ liệu chuẩn hóa và $S = \frac{1}{N} \hat{X} \hat{X}^T$ là ma trận hiệp phương sai. S là ma trận bán xác định dương.

Với K bằng 0, ta có:

$$L = \sum_{i=1}^{D} u_i^T S u_i = \frac{1}{N} \|\hat{X}^T U\|_F^2 = \frac{1}{N} trace(\hat{X}^T U U^T \hat{X})$$

$$= \frac{1}{N} trace(\hat{X}^T \hat{X}) = \frac{1}{N} trace(\hat{X} \hat{X}^T) = trace(S) = \sum_{i=1}^{D} \lambda_i \quad (6)$$

Ở đây $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_i$ là các trị riêng của ma trận xác định dương S. Chú ý rằng các giá trị riêng này là các phần tử không âm.

Như vậy L không phụ thuộc vào hệ cở sở trực giao U và bằng tổng các phần tử trên đường chéo của S. Nói cách khác, L chính là tổng các phương sai của từng thành phần dữ liệu ban đầu.

Vì vậy, hàm mát J được cho bởi (5) tương đương với việc tối đa:

$$F = L - J = \sum_{i=1}^{K} u_i^T S u_i$$

Định lý 1:F đạt giá trị lớn nhất bằng $\sum_{i=1}^{K} \lambda_i$ khi u_i là các vector riêng u_i ứng với giá trị riêng λ_i lập thành một hệ trực chuẩn: u_i trực giao và $\|u_i\|_2 = 1$, $i = 1, ..., k$

Các bước PCA

1. Tính kỳ vọng của toàn bộ dữ liệu

$$\hat{x} = \frac{1}{N} \sum_{n=1}^{N} x_n$$

2. Tính dữ liệu chuẩn hóa $\hat{x}_n$

$$\hat{x}_n = x_n - \bar{x} \quad n = 1, 2, ..., N.$$

3. Tính ma trận hiệp phương sai

$$S = \frac{1}{N} \hat{X} \hat{X}^T$$

4. Tính các giá trị riêng λ_i và vector riêng u_i có $\|u_i\|_2 = 1$ của ma trận này, sắp xếp chúng theo thứ tự giảm dần của giá trị riêng.

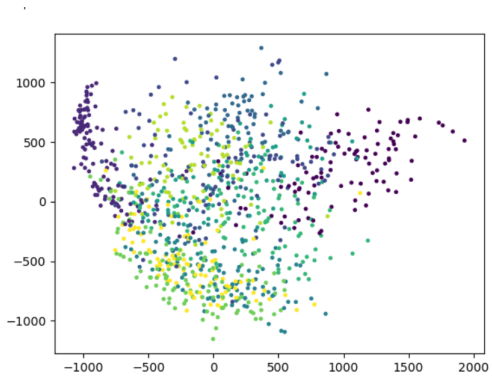
5. Chọn k giá trị riêng lớn nhất và k vector riêng trực chuẩn tương ứng. Xây dựng ma trận U_k

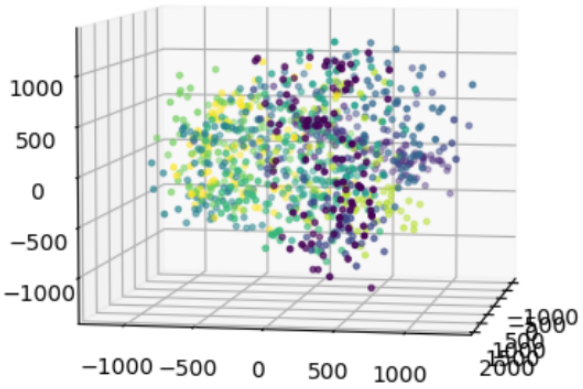
6. Các cột của $U_{i\,i=1}^{k}$ là hệ cơ sở trực chuẩn, tạo thành không gian con của k thành phần chính, gần với phân bố của dữ liệu ban đầu đã chuẩn hóa.

7. Chiếu dữ liệu ban đầu đã chuẩn hóa $\hat{X}$ xuống không gian con nói trên. Dữ liệu mới chính là tọa độ của các điểm dữ liệu trên không gian mới

$$Z = U_k^T \hat{X}$$

2.1.2 Kết quả sau khi giảm chiểu để visualize 2D và 3D



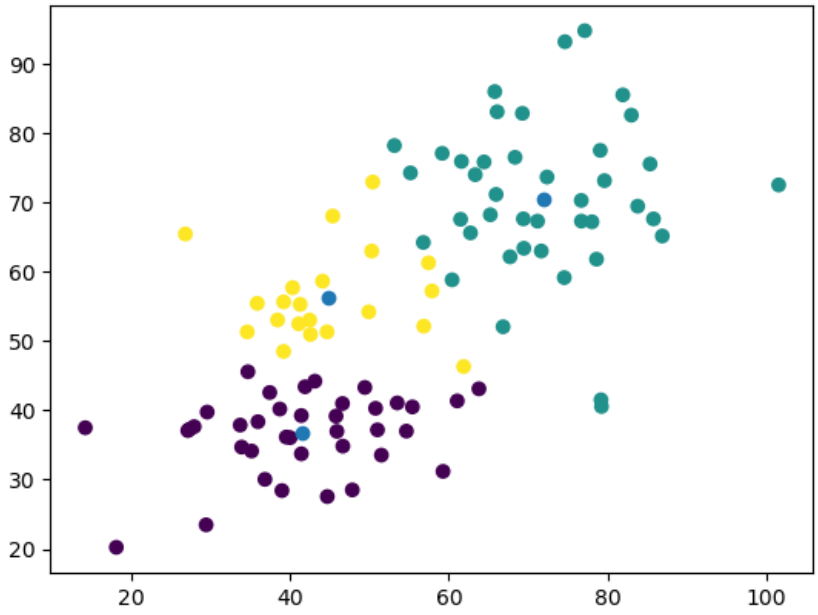


2.2 Xây dựng mô hình bằng phương pháp K-Means Clustering

Thuật toán K Means là một trong những thuật toán cơ bản của nhóm các thuật toán Unsupervised Learning. Trong thuật toán K Means clustering, chúng ta không biết nhãn (labels) của từng điểm dữ liệu. Mục đích của các thuật toán phân cụm nói chung và K Means nói riêng là làm thế nào để phân dữ liệu thành các cụm có (cluster) khác nhau sao cho dữ liệu trong mỗi cụm có cùng một tính chất nào đó giống nhau.

Ý tưởng đơn giản nhất về cụm là tập hợp những điểm nằm ở gần nhau trong một không gian nào đó. Không gian có thể có nhiều chiều khi dữ liệu rất lớn.

Hình dưới minh họa về ba cụm dữ liệu



Mỗi điểm hiển thị màu xanh dương biểu thị cho tâm cụm.

2.2.1 Tóm tắt cơ sở lý thuyết và thuật toán phân cụm K-Means

Chúng ta sẽ gọi như sau

$$X = [x_1, x_2, x_3, ...x_N] \in R^{d \times N}$$

$$M = [m_1, m_2, ...m_k] \in R^{d \times 1}$$

Với mỗi x_i được phân vào cluster k thì ta có ràng buộc

$$y_{ik} = 1; \quad y_{ij} = 0; \qquad j \neq k$$

Ràng buộc của của y_i có thể được viết dưới dạng toán học như sau

$$y_{ik} \in \{0,1\}, \quad \sum_{k=1}^{k} y_{ik} = 1 \qquad (1)$$

m_k là trung tâm của mỗi cụm cluster, nếu tất cả các cụm được phân vào cluster bởi m_k thì, mỗi điểm dữ liệu x_i khi phân vào cluster k thì sẽ có sai số là $(x_i - m_k)$.

Vậy, sai số của một điểm dữ liệu x_i là:

$$\|x_i - m_k\|_2^2 = y_{ik}\|x_i - m_k\|_2^2 = \sum_{j=1}^{K} y_{ij}\|x_i - m_k\|_2^2$$

Như vậy, sai số của toàn bộ điểm dữ liệu là:

$$L(Y,M) = \sum_{i=1}^{N}\sum_{j=1}^{K} \|x_i - m_k\|_2^2$$

Trong đó $Y = [y_1, y_2, y_3, ...y_N]$ và $M = [m_1, m_2, ...m_k]$

Như vậy, chúng ta cần tối ưu bài toán với điều kiện ở (1):

$$Y, M = \arg\min_{Y,M} \sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} \left\|x_i - m_j\right\|_2^2 \qquad (2)$$

$$s.t : y_{ij} \in \{0,1\} \quad \forall i,j; \qquad \sum_{j=1}^{K} y_{ij} = 1 \quad \forall i$$

Để giải bài toán này, ta có thể tiếp cận theo hướng đơn giản. Đó xen kẽ giải Y và M khi biến còn lại được cố định.

Như vậy, ta sẽ giải xen kẽ 2 bài toán con như sau:

Cố định M, tìm Y

Mục đích của bài toán này là tìm được bộ các label vector để hàm mất mát đạt giá trị nhỏ nhất. Điều này có nghĩa là ta sẽ tìm cụm cho mỗi điểm dữ liệu.

Bài toán (2) được chia nhỏ thành bài toán tìm label vector cho một điểm dữ liệu i như sau:

$$y_i = argmin \sum_{j=1}^{K} y_{ij} \|x_i - m_j\|_2^2 \qquad (3) s.t : y_{ij} \in \{0,1\}; \qquad \sum_{j=1}^{k} y_{ij} = 1$$

Ta có thể viết lại bài toán (3) như sau:

$$j = argmin \|x_i - m_j\|_2^2$$

Điều này có thể hiểu rằng ta đang cố gắng tìm tâm cụm gần với điểm x_i nhất. Với $\|x_i - m_j\|$ là khoảng cách từ x_i đến tâm m_j

Cố định Y, tìm M

Mục đích của bài toán này là tìm được các tâm cụm mới của mỗi cụm sau khi tìm được các label vector sao cho hàm mất mát đạt giá trị nhỏ nhất.

Sau khi có được các label vector bài toán (2) được rút gọn thành bài toán tìm center cho một cluster như sau:

$$m_j = argmin \sum_{i=1}^{N} y_{ij} \|x_i - m_j\|_2^2 \qquad (4)$$

Bài toán này giải nó bằng đạo hàm theo ẩn m_j. Để thuận tiện, chúng ta đặt $g(m_j) = \sum_{i=1}^{N} y_{ij} \|x_i - m_j\|_2^2$ hàm bên trong dấu *argmin*. Ta sẽ có đạo hàm như sau:

$$\frac{\partial g(m_j)}{\partial m_j} = 2 \sum_{i=1}^{N} y_{ij}(x_i - m_j) = 0$$

$$-> 2\sum_{i=1}^{N}(y_{ij}x_i - y_{ij}m_j) = 0$$

$$-> \sum_{i=1}^{N} y_{ij}x_i = \sum_{i=1}^{N} y_{ij}m_j$$

Như vậy, rút m_j ra, ta được:

$$m_j = \frac{\sum_{i=1}^{N} y_{ij}x_i}{\sum_{i=1}^{N} y_{ij}}$$

Thuật toán như sau [Nguồn: link https://machinelearningcoban.com/2017/01/01/kmeans/]

Đầu vào: Dữ liệu X và số lượng cluster cần tìm K.

Đầu ra: Các center M và label vector cho từng điểm dữ liệu

1. Chọn K điểm bất kỳ làm các center ban đầu

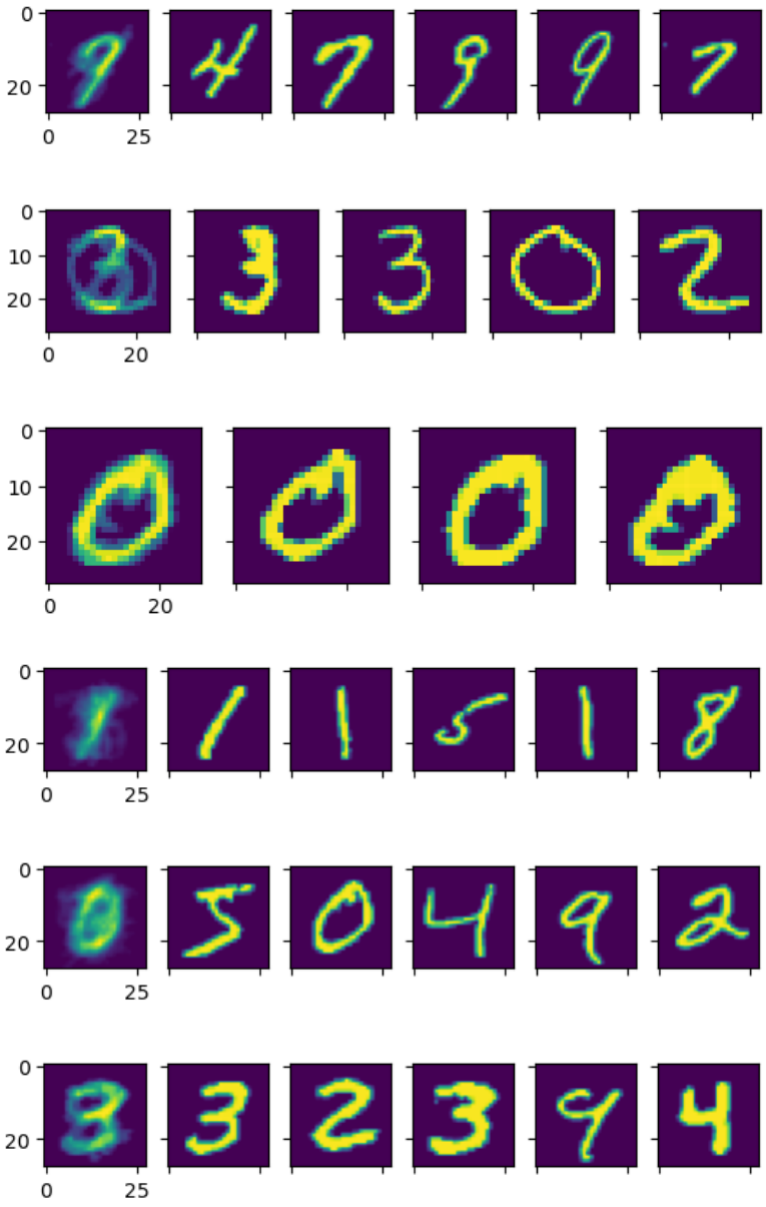
2. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất

3. Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi nhiều thì ta dừng thuật toán

4. Cập nhật các center cho từng cluster bằng cách lấy trung bình cộng của các điểm dữ liệu đã được gán vào cluster đó sau bước 2.

5. Quay lại bước hai

2.2.2 Kết quả sau khi phân cụm



3 Mô hình dự đoán cho bộ dữ liệu MNIST

Trong phần này, chúng em sẽ trình bày cách triển khai và áp dụng hai mô hình khá phổ biến cho bài toán phân loại nhiều lớp, đó là Multinomial Logistic Regression và ANN. Trước khi đi vào các mô hình chi tiết, chúng em xin đưa ra một số quy ước về mặt ký hiệu khi trình bày:

X: Dữ liệu đầu vào.

y: Nhãn đầu ra.

$x^{(i)}$: Mẫu dữ liệu thứ i, nếu dữ liệu X có dạng bảng thì còn có thể gọi là dữ liệu hàng thứ i.

$y^{(i)}$: Nhãn thứ i, ứng với mẫu dữ liệu $x^{(i)}$.

x_j: Thuộc tính j của dữ liệu đầu vào.

y_j: Thuộc tính j của nhãn đầu ra.

Trước khi đưa vào mô hình, dữ liệu X sẽ được giảm số chiều bằng phương pháp PCA đã được trình bày ở trên, do đó, với mỗi mẫu dữ liệu $x^{(i)}$ sẽ có dạng đầy đủ là:

$$x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_{100}^{(i)})^T$$

Cùng với đó, trước khi đưa vào xây dựng mô hình, các dữ liệu đầu vào ở mỗi thuộc tính sẽ được chuẩn hoá để nằm trong đoạn $[0, 1]$, qua công thức:

$$x_j = \frac{x_j - min(x_j)}{max(x_j) - min(x_j)}$$

3.1 Xây dựng mô hình Multi-softmax

Mô hình Multinomial Logistic Regression, hay còn gọi là Softmax Regression, là một trong những phương pháp phân loại phổ biến trong Machine Learning. Nó là một phương pháp mở rộng của mô hình Logistic Regression truyền thống, thuộc kiểu one-vs-rest cho phép dự đoán đầu ra trong nhiều hơn hai lớp.

Mô hình Softmax sử dụng hàm Softmax để biến đổi các giá trị đầu vào thành một phân phối xác suất trên các lớp đầu ra. Sau khi có được phân phối xác suất, mô hình sử dụng một

hàm mục tiêu để tính toán lại các trọng số nhằm mục đích gia tăng độ chính xác của dự đoán với các giá trị đầu ra thực tế. Hàm Softmax giúp chuẩn hóa giá trị đầu vào để chúng có thể được hiểu như là xác suất của các lớp đầu ra tương ứng. Về ứng dụng, mô hình Multinomial Logistic Regression (Softmax) được sử dụng trong nhiều lĩnh vực khác nhau, bao gồm nhận dạng ảnh, phân loại văn bản, và dự đoán hành vi khách hàng. Nó là một trong những phương pháp phân loại quan trọng trong Machine Learning, và đóng vai trò quan trọng trong việc xử lý dữ liệu đa lớp.

Trong bài báo cáo này, do thời lượng của báo cáo, bọn em xin phép trình bày ngắn gọn các bước triển khai thuật toán (kể cả đối với mô hình ANN ở phần sau) nhưng vẫn đảm bảo thể hiện đầy đủ ý tưởng của nó.

Trước tiên, ta có các quy ước:

- N và D lần lượt là số hàng dữ liệu và số thuộc tính

- $x_j^{(i)}$ có kích cỡ $(100, 1)$

- $y^{(i)}$ sẽ được chuẩn hoá về dạng one-hot vector có cỡ là $(10, 1)$

- W: Ma trận trọng số weights cho bài toán của chúng ta, có cỡ $(10, 100)$, với 10 lấy từ số lượng nhãn trong mô hình và 100 là số chiều ban đầu của một mẫu dữ liệu đầu vào.

- b: Vector bias cho bài toán của chúng ta, có cỡ $(10, 1)$

- $\hat{y}^{(i)}$: Vector xác suất được tính từ hàm softmax, có cỡ $(10, 1)$ tương tự với biến đầu ra

- ∂W: Ma trận đạo hàm theo hàm mục tiêu J đối với trọng số weights, có cỡ tương tự W là $(10, 100)$

- ∂b: Vector đạo hàm theo hàm mục tiêu J đối với bias, có cỡ tương tự b là $(10, 1)$

Ta có các bước để training mô hình Softmax cho dữ liệu như sau:

1. Khởi tạo các giá trị W và b

2. Thực hiện vòng lặp với các $epoch := 1, 2, ...$:

    2.1. For $i := 1, 2, 3, ...N$:

    $$\hat{y}^{(i)} = softmax(Wx^{(i)} + b)$$

Hàm tổn thất tại dữ liệu thứ (i): $loss(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)})^T \log y^{(i)}$

2.2. Hàm mục tiêu:

$$J = \frac{1}{N} \sum_{i=1}^{N} loss(\hat{y}^{(i)}, y^{(i)})$$

2.3. Ta có:

$$\partial W = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)})(x^{(i)})^T$$

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

2.4. Cập nhật W và b qua Gradient descent với hệ số học (learning rate) α:

$$W := W - \alpha \partial W$$

$$b := b - \alpha \partial b$$

Trong mô hình mà chúng em đã triển khai, hệ số học được chọn là 0.8 và được thực hiện qua 500 bước lặp.

3.2 Xây dựng mô hình ANN

Mô hình ANN (Artificial Neural Network) là một mô hình học máy được lấy cảm hứng từ cấu trúc và hoạt động của hệ thống thần kinh của con người. Mô hình này là một hệ thống liên kết các đơn vị tính toán (neuron) với nhau để thực hiện các tác vụ học máy. Mỗi neuron trong ANN nhận đầu vào từ các neuron khác hoặc từ các giá trị đầu vào và tính toán đầu ra dựa trên các trọng số kết nối và hàm kích hoạt. Các trọng số kết nối giữa các neuron được điều chỉnh trong quá trình huấn luyện để tối ưu hóa đầu ra của mô hình.

Mô hình ANN có thể được áp dụng cho nhiều loại dữ liệu khác nhau, bao gồm dữ liệu ảnh, âm thanh, văn bản, và nhiều loại dữ liệu khác. Điều này giúp cho ANN trở thành một công cụ hữu ích trong nhiều lĩnh vực, ví dụ như xử lý ảnh, phân loại ảnh, xử lý ngôn ngữ tự nhiên.

Tuy nhiên, mô hình ANN cũng có những hạn chế nhất định, bao gồm khả năng bị quá khớp dữ liệu, thời gian huấn luyện lâu, và khó khăn trong việc giải thích quá trình ra quyết định của mô hình. Tuy nhiên, với sự phát triển của công nghệ và các phương pháp tối ưu hóa mới, ANN vẫn là một mô hình học máy phổ biến và mạnh mẽ được sử dụng rộng rãi trong thực tế.

Cũng giống như mô hình Softmax Regression, trước khi bước vào quá trình triển khai thuật toán huấn luyện mô hình ANN, ta có các quy ước như sau:

- N và D lần lượt là số hàng dữ liệu và số thuộc tính, (với bài toán của chúng ta $N = 60000$, $D = 100$.

- N_{batch} là kích cỡ trong một mini-batch

- $x_j^{(i)}$: Là mẫu dữ liệu x đầu vào, với i là chỉ số hàng (chỉ số một mẫu dữ liệu hay một example), j là chỉ số thuộc tính, ta quy ước với một mẫu $x^{(i)}$ là có kích cỡ $(100, 1)$.

- L: là số lượng các layer của kiến trúc mạng Neural, tính từ hidden layer thứ nhất cho đến output layer.

- $n^{[l]}$: là số lượng các unit trong layer thứ l, với $l = 0, 1, 2, ..., L$ ($l = 0$ chính là input layer.

- $y^{(i)}$: Là nhãn y đầu ra, với i là chỉ số hàng (chỉ số một mẫu dữ liệu hay một example), ở đây, ta sẽ chuẩn hoá các nhãn đầu ra về dạng one-hot vector có cỡ là $(10, 1)$

- $W^{[l]}$: Ma trận trọng số weights đối với layer thứ l, ở đây $l = 1, 2, 3, ..., L$, ta quy ước các ma trận trọng số này có kích cỡ $(n^{[l]}, n^{[l-1]})$. Khi đó đối với bài toán của chúng ta, kích cỡ của $W^{[1]}$ và $W^{[L]}$ lần lượt là $(n^{[1]}, 100)$ và $(10, n^{[L-1]})$

- $b^{[l]}$: Vector bias đối với layer thứ l, tương tự $l = 1, 2, 3, ..., L$, ta quy ước các vector này có kích cỡ $(n^{[l]}, 1)$

- $z^{[l]}$: Hàm biến đổi tuyến tính ở layer thứ l, $(l = 1, 2, 3, ..., L)$, hàm này là một vector hàm , ta quy ước các vector này có kích cỡ $(n^{[l]}, 1)$

- $a^{[l]}$: Hàm kích hoạt ở layer thứ l, $(l = 0, 2, 3, ..., L)$, Ở đây, $a^{[0]} = x^{(i)}$, với các giá trị $l > 0$, hàm này có cùng kích cỡ với $z^{[l]}$ kích cỡ $(n^{[l]}, 1)$, đối với bài toán của chúng ta, từ $l = 1, 2, 3, ..., L-1$, ta sử dụng hàm kích hoạt $ReLU$, với layer cuối cùng thì hàm kích hoạt sẽ là $Softmax$

- $\partial W^{[l]}$: Ma trận đạo hàm theo hàm mục tiêu đối với trọng số weights ở layer thứ l, có cỡ

tương tự $W^{[l]}$ là $(n^{[l]}, n^{[l-1]})$

- $\partial b^{[l]}$: Vector đạo hàm theo hàm mục tiêu đối với bias ở layer thứ l, có cỡ tương tự $b^{[l]}$ là $(n^{[l]}, 1)$

- $\partial z^{[l]}$: vector đạo hàm theo hàm mục tiêu đối với $z^{[l]}$, có cỡ tương tự $z^{[l]}$ là $(n^{[l]}, 1)$

- $\partial W^{[l](i)}$: Ma trận đạo hàm theo hàm tổn thất đối với trọng số weights ở layer thứ l, ở dữ liệu thứ i.

- $\partial b^{[l](i)}$: Vector đạo hàm theo hàm tổn thất đối với bias ở layer thứ l, ở dữ liệu thứ i.

- $\partial z^{[l](i)}$: Vector đạo hàm theo hàm tổn thất đối với $z^{[l]}$, ở dữ liệu thứ i.

Ta có các bước để training mô hình mạng Neural nhân tạo với L layer cho dữ liệu MNIST như sau:

1. Khởi tạo các giá trị $W^{[l]}$ và $b^{[l]}$. Ở đây để giảm bớt hiện tượng vanishing gradient trong quá trình training, ta sẽ sử dụng cách khởi tạo He (Hay còn tên gọi khác là Kaiming). Với mỗi phần tử trong ma trận $W^{[l]}$ sẽ được khởi tạo tuân theo phân bố chuẩn $\mathcal{N}(0, \frac{2}{n^{[l-1]}})$. Đối với $b^{[l]}$ có thể khởi tạo theo cách này hoặc khởi tạo ngẫu nhiên các giá trị bằng 0.

2. For $epoch := 1, 2, ...$:

Chúng ta sẽ chia bộ dữ liệu training thành các mini-batch và train trên mỗi mini-batch , do đó trong một epoch lúc này sẽ trải qua một số lượng lần cập nhật tham số, số này được lấy bằng phần nguyên cận trên của phép chia của N cho N_{batch} ta gọi nó với cái tên steps.

For step in steps:

2.1 For $i := 1, 2, 3, ...N_{batch}$:

Forward propagation:

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

$$a^{[1]} = ReLU(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[1]}$$

$$a^{[2]} = ReLU(z^{[2]})$$

$$\vdots$$

$$z^{[L-1]} = W^{[L-1]}a^{[L-2])} + b^{[L-1]}$$

$$a^{[L-1]} = ReLU(z^{[L-1]})$$

$$z^{[L]} = W^{[L]}a^{[L-1]} + b^{[L]}$$

$$a^{[L]} = Softmax(z^{[L]})$$

Hàm tổn thất tại dữ liệu thứ (i): $L(a^{[L]}, y^{(i)}) = -(y^{(i)})^T \log a^{[L]}$

Backward propagation:

$$\partial z^{[L](i)} = a^{[L]} - y^{(i)}$$

$$\partial W^{[L](i)} = \partial z^{[L](i)}(a^{[L-1]})^T$$

$$\partial b^{[L](i)} = \partial z^{[L](i)}$$

$$\partial z^{[L-1](i)} = (W^{[L]})^T \partial z^{[L](i)} \odot I\{z^{[L-1](i)} > \vec{0}\}$$ (Đạo hàm của hàm $ReLU$)

$$\partial W^{[L-1](i)} = \partial z^{[L-1](i)}(a^{[L-2]})^T$$

$$\partial b^{[L-1](i)} = \partial z^{[L-1](i)}$$

$$\vdots$$

$$\partial z^{[1](i)} = (W^{[2]})^T \partial z^{[2](i)} \odot I\{z^{[1](i)} > \vec{0}\}$$

$$\partial W^{[1](i)} = \partial z^{[1](i)}(a^{[0]})^T$$

$$\partial b^{[1](i)} = \partial z^{[1](i)}$$

2.2 Hàm mục tiêu:

$$J = \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} L(a^{[L]}, y^{(i)})$$

2.3 Ta có: $\partial W^{[l]} = \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \partial W^{[l](i)}$

$$\partial b^{[l]} = \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \partial b^{[l](i)}$$

2.4 Để cập nhật W và b, chúng ta sẽ sử dụng thuật toán RMSProp là một thuật toán cải biên của Gradient descent:

$SW^{[l]} := \beta SW^{[l]} - (1-\beta)\partial W^{[l]} \odot \partial W^{[l]}$ (Giá trị β thường được lựa chọn là 0.999)

$$Sb^{[l]} := \beta Sb^{[l]} - (1-\beta)\partial b^{[l]} \odot \partial b^{[l]}$$

$$W^{[l]} := W^{[l]} - \alpha \partial W^{[l]} \sqrt{SW^{[l]} + \varepsilon}$$

$$b^{[l]} := b^{[l]} - \alpha \partial b^{[l]} \sqrt{Sb^{[l]} + \varepsilon}$$

Theo các công thức trên thì RMSProp tính toán trung bình bình phương của gradient của mỗi tham số trong quá trình tối ưu hóa, và sử dụng giá trị này để điều chỉnh tỷ lệ học cho từng tham số. Điều này giúp giảm đáng kể sự dao động của các tham số trong quá trình tối ưu hóa, đồng thời tăng tốc quá trình hội tụ đến giá trị tối ưu.

Trong mô hình mà chúng tôi lựa chọn, có 2 hidden layer với số units lần lượt là 128, 64, hệ số học là 10^{-5}, kích cỡ N_{batch} là 32 và train trong 50 epochs

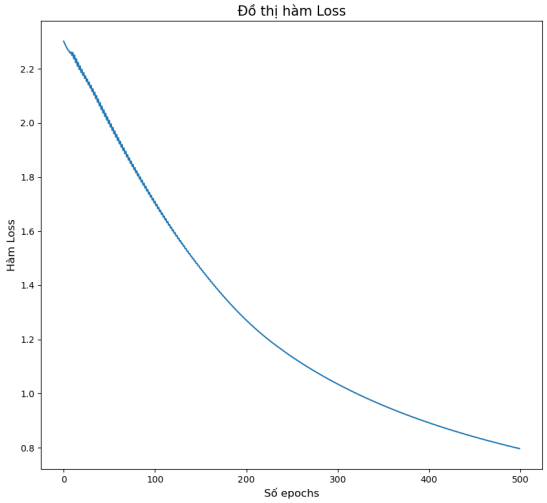
3.3 Đánh giá các mô hình

Để đánh giá mô hình, ta sử dụng các thông số về *accuracy*, *precision*, *recall* và $F1-score$

Với các chỉ số về *precision* và *recall*, với mỗi một nhãn C, ta tính *precision* và *recall* tương ứng cho các lớp đó, đối với chỉ số $F1-score$ ta tính trung bình các chỉ số *precision* và *recall* và $F1-score = 2PR/(P+R)$

3.3.1 Mô hình Multinomial Logistic Regression

Ta có đồ thị của hàm mục tiêu:



Có thể thấy với các siêu tham số lựa chọn hàm mục tiêu có mức giảm rất ổn định, và ta có thể gia tăng thêm số epochs để tiếp tục giảm hàm mục tiêu vì đồ thị trên cũng chưa cho thấy xu hướng hội tụ của nó. Tiếp theo ta có các thông số:

Training evaluation:

1. Accuracy: 0.8666
2. Precision:

Label:	Precision:
0	0.9316
1	0.8393
2	0.8943
3	0.8326
4	0.8621
5	0.856
6	0.8992
7	0.8893
8	0.8454
9	0.8215

3. Recall:

Label:	Recall:
0	0.945
1	0.9644
2	0.8226
3	0.8348
4	0.8851
5	0.7532
6	0.9287
7	0.886
8	0.786
9	0.8363

4. F1-score: 0.8646

Validation evaluation:

1. Accuracy: 0.8779
2. Precision:

Label:	Precision:
0	0.9282
1	0.8683
2	0.9097
3	0.8487
4	0.8626
5	0.8942
6	0.8962
7	0.8848
8	0.848
9	0.8466

3. Recall:

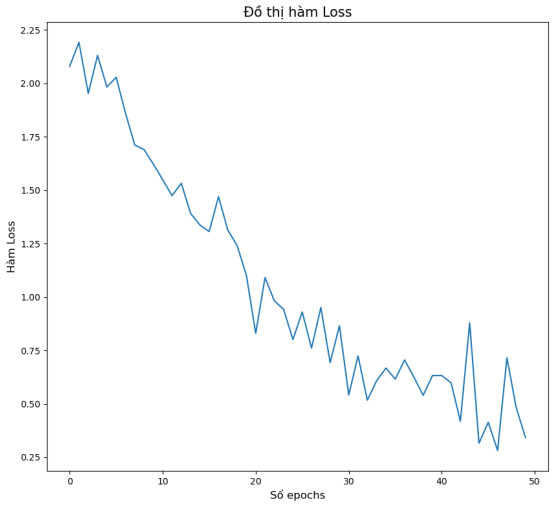
Label:	Recall:
0	0.9633
1	0.9639
2	0.8198
3	0.8772
4	0.9012
5	0.7578
6	0.928
7	0.8813
8	0.8131
9	0.8533

4. F1-score: 0.8761

Các chỉ số ở mức ổn. Thêm một lý do nữa để gia tăng thêm số epochs ta thấy *accuracy* của tập validation đang nhỉnh hơn chút so với tập training.

3.3.2 Mô hình ANN

Ta có đồ thị của hàm mục tiêu:



Rõ ràng so với mô hình Softmax, hàm mục tiêu của ANN có những dao động không ổn định tại các thời điểm nhưng xu hướng chung vẫn là giảm. Và các thông số đánh giá như sau:

Training evaluation:

1. Accuracy: 0.8638
2. Precision:

Label:	Precision:
0	0.935
1	0.8976
2	0.8733
3	0.8703
4	0.8615
5	0.8006
6	0.8927
7	0.8791
8	0.7869
9	0.8293

3. Recall:

Label:	Recall:
0	0.935
1	0.9465
2	0.8377
3	0.7981
4	0.8795
5	0.8094
6	0.9093
7	0.875
8	0.7988
9	0.8339

4. F1-score: 0.8622

Validation evaluation:

1. Accuracy: 0.871
2. Precision:

Label:	Precision:
0	0.9181
1	0.9294
2	0.8847
3	0.8543
4	0.8637
5	0.8075
6	0.9016
7	0.8669
8	0.803
9	0.8634

3. Recall:

Label:	Recall:
0	0.949
1	0.9515
2	0.8479
3	0.8188
4	0.9033
5	0.7948
6	0.8987
7	0.8745
8	0.8162
9	0.8394

4. F1-score: 0.8692

Với các siêu tham số đã chọn, các chỉ số mô hình ANN cũng ở mức khá tốt.

4 Kết luận

Multinomial Logistic Regression và Artificial Neural Networks (ANN) là hai mô hình quan trọng trong lĩnh vực học máy. Mỗi mô hình có ưu điểm và nhược điểm riêng, và được sử dụng phù hợp với các bài toán và tập dữ liệu khác nhau.

Đối với bài toán đã cho, hai mô hình đều tỏ ra khá hiệu quả khi đưa ra các kết quả ở mức khá tốt. Tuy nhiên, phần nào có thể thấy, Multinomial Logistic Regression cần phải được training nhiều hơn thì mới có một sự tương đồng trong kết quả ở các thông số đánh giá như là ANN. Điều này cũng hoàn toàn dễ hiểu khi ANN là một mô hình phức tạp hơn, có khả năng học và mô hình hóa các mối quan hệ phi tuyến giữa các đặc trưng. ANN thường được sử dụng trong các bài toán phân loại và dự đoán trong các tập dữ liệu lớn và phức tạp.

Tài liệu tham khảo

[1] Vũ Hữu Tiệp, *Machine Learning cơ bản*, https://machinelearningcoban.com/.

[2] Ian Goodfellow and Yoshua Bengio and Aaron Courville *Deep Learning*, MIT Press, http://www.deeplearningbook.org, 2016.

[3] Andrew Ng's *Machine Learning Specialization*, Standford University, https://www.coursera.org/specializations/machine-learning-introduction.

[4] Andrew Ng's *Deep Learning Specialization*, Standford University, https://www.coursera.org/specializations/deep-learning.