TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Lê Xuân Hoàng - Lê Xuân Huy

HỌC CÓ GIÁM SÁT VỚI DỮ LIỆU NHIỄU BẰNG PHƯƠNG PHÁP LỰA CHỌN DỮ LIỆU

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN
CHƯƠNG TRÌNH CHÍNH QUY

Tp. Hồ Chí Minh, tháng 07/2024

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Lê Xuân Hoàng - 20120089

Lê Xuân Huy - 20120494

HỌC CÓ GIÁM SÁT VỚI DỮ LIỆU NHIỄU BẰNG PHƯƠNG PHÁP LỰA CHỌN DỮ LIỆU

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN
CHƯƠNG TRÌNH CHÍNH QUY

GIÁO VIÊN HƯỚNG DẪN

TS. Nguyễn Ngọc Thảo

ThS. Trần Trung Kiên

Tp. Hồ Chí Minh, tháng 07/2024

Lời cảm ơn

Chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến ThS. Trần Trung Kiên và TS. Nguyễn Ngọc Thảo, những người đã tận tình hướng dẫn, góp ý chỉ bảo cho chúng em trong suốt cả đề tài. Sự quan tâm, theo dõi của thầy cô đã giúp chúng em vượt qua nhiều khó khăn để hoàn thành khóa luận này.

Chúng em xin cảm ơn Ban giám hiệu nhà trường, các thầy cô của Khoa Công Nghệ Thông Tin đã cung cấp cho chúng em những nền tảng kiến thức vững chắc. Sự ân cần và tâm huyết của thầy cô không chỉ cung cấp những kiến thức chuyên môn sâu rộng mà còn có những kỹ năng thực tiễn.

Chúng em xin cảm ơn gia đình, bạn bè, cùng những người đã luôn ở bên cạnh chúng em để động viên, quan tâm, cổ vũ để chúng em vượt qua khó khăn, thử thách trong quá trình thực hiện khóa luận.

Tp.Hồ Chí Minh, tháng 7/2024

Lê Xuân Hoàng

Lê Xuân Huy

Mục lục

Danh sách hình

Danh sách bảng

Chương 1

Giới thiệu

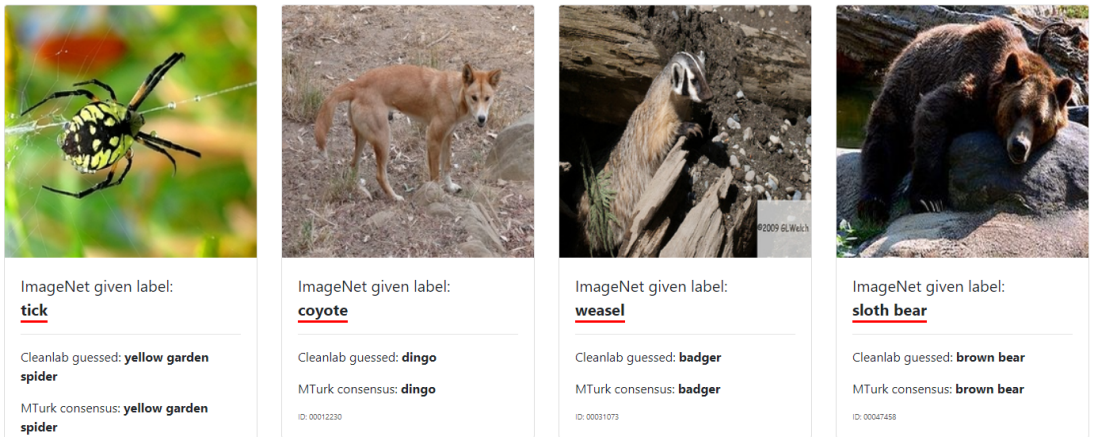
1.1 Đặt vấn đề

Trong thập kỷ qua, lĩnh vực học máy đã chứng kiến sự phát triển vượt bậc, với nhiều thành tựu đáng kể trong việc huấn luyện các mô hình dự đoán và phân loại. Các mô hình mạng nơ-ron, đặc biệt là trong học có giám sát, đã được ứng dụng rộng rãi trong nhiều lĩnh vực như y tế, tài chính, thị giác máy tính và xử lý ngôn ngữ tự nhiên. Những tiến bộ này phần lớn dựa trên sự sẵn có của dữ liệu lớn, khả năng tính toán mạnh mẽ và sự xuất hiện của những mô hình mạng nơ-ron tiên tiến.

Chất lượng dữ liệu vẫn luôn là yếu tố then chốt quyết định hiệu suất của các mô hình. Trong thực tế, dữ liệu thu thập thường chứa nhiều nhiễu, gây ra những thách thức đáng kể trong quá trình huấn luyện. Dữ liệu nhiễu là những điểm dữ liệu không phản ánh đúng bản chất thực sự của hiện tượng được nghiên cứu. Việc thu thập và xử lý dữ liệu đang trở thành một công việc phức tạp và tốn kém.

Các tập dữ liệu lớn như ImageNet cho thị giác máy tính hay các tập dữ liệu y tế, tài chính đều đối mặt với vấn đề dữ liệu nhiễu. Hình 1.1 minh họa cho điều này. Cụ thể hơn, nghiên cứu của Northcutt et al.(2021)[11] chỉ ra rằng tỷ lệ nhãn nhiễu trong tập dữ liệu ImageNet là khoảng 6%. Một số phân tích sâu hơn cho thấy tỷ lệ nhãn nhiễu trên tập dữ liệu ImageNet có thể cao hơn. Ước tính tỷ lệ nhãn nhiễu có thể gần 20% khi bao gồm cả những nhãn nhiễu không được phát hiện. Dữ liệu nhiễu có thể chiếm một tỷ lệ đáng kể cho quá trình huấn luyện mô hình. Điều này đặc biệt đúng với các dữ liệu được thu thập tự động trên internet, nơi mà chất lượng dữ liệu khó kiểm soát một cách chính xác.

Dữ liệu nhiễu có thể ảnh hưởng tiêu cực đến hiệu suất của mô hình theo nhiều cách khác nhau. Ví dụ, nó có thể dẫn đến việc học sai mô hình,



Hình 1.1: Một số mẫu dữ liệu bị gán nhãn sai trên tập dữ liệu ImageNet.[2]

giảm độ chính xác của dự đoán, tăng nguy cơ sai sót và ảnh hưởng đến khả năng khái quát hóa của mô hình. Do đó, việc giải quyết bài toán học có giám sát với dữ liệu nhiễu là một vấn đề đáng quan tâm. Đây là một lĩnh vực nghiên cứu đầy tiềm năng và đáng được đầu tư công sức.

Dữ liệu nhiễu bao gồm nhiều loại khác nhau, có thể được phân loại theo các hình thức như:

- **Dữ liệu có nhãn nhiễu:** loại dữ liệu này xảy ra khi nhãn của một điểm dữ liệu không chính xác hoặc không phản ánh đúng bản chất thực sự của nó.
- **Dữ liệu không đầy đủ:** loại dữ liệu này thiếu một số giá trị thuộc tính cho một số mẫu dữ liệu.
- **Dữ liệu không nhất quán:** loại dữ liệu này có thể chứa các giá trị thuộc tính mâu thuẫn hoặc không hợp lý.
- **Dữ liệu ngoại lệ:** loại dữ liệu này bao gồm những điểm dữ liệu khác biệt đáng kể so với phần còn lại của dữ liệu.
- **Dữ liệu nhiễu ngẫu nhiên:** loại dữ liệu này do các yếu tố ngẫu nhiên gây ra, chẳng hạn như lỗi cảm biến hoặc nhiễu trong quá trình truyền tải dữ liệu.

Trong khóa luận này, chúng em chỉ tập trung vào dữ liệu có nhãn nhiễu.

[2]Nguồn: labelerrors.com

1.2 Bài toán học có giám sát với dữ liệu nhiễu

Định nghĩa bài toán

Bài toán học có giám sát với dữ liệu có nhãn nhiễu được phát biểu như sau:

- Cho tập dữ liệu có chứa nhiễu $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$, trong đó (x_i, y_i) lần lượt là input và nhãn của mẫu thứ i. Chúng ta chỉ quan sát được input và nhãn nhiễu của nó $\{y_i\}_{i=1}^n$ mà không quan sát được nhãn thật của nó $\{\widetilde{y}_i\}_{i=1}^n$.
- Yêu cầu: tìm hàm $f : \mathbb{R}^d \to \mathbb{R}$ có khả năng dự đoán nhãn thật $\widetilde{y} \in \mathbb{R}$ của một input mới $x \in \mathbb{R}^d$ nằm ngoài $\mathcal{D}$.

Khó khăn của bài toán

Bài toán học có giám sát với dữ liệu nhiễu là bài toán thiết yếu trong thời buổi dữ liệu được ví như là "mỏ vàng". Đây không phải là một nhiệm vụ đơn giản. Trước hết, việc xác định đâu là dữ liệu nhiễu và đâu là dữ liệu sạch là một vấn đề khó khăn, ảnh hưởng trực tiếp đến hiệu quả của mô hình. Việc loại bỏ sai dữ liệu nhiễu có thể dẫn đến mất thông tin quan trọng, trong khi giữ lại dữ liệu nhiễu có thể khiến mô hình học theo các thông tin sai lệch và giảm độ chính xác dự đoán. Thứ hai, các mô hình học máy phức tạp như mạng nơ-ron có khả năng ghi nhớ bất kỳ nhãn nào (thậm chí ngẫu nhiên) của dữ liệu[15]. Do đó, nếu không có các biện pháp xử lý phù hợp, mô hình có thể học theo các thông tin sai lệch từ dữ liệu nhiễu và giảm khả năng tổng quát hóa trên dữ liệu mới.

Ý nghĩa của bài toán

Việc giải quyết được bài toán này sẽ mang lại nhiều giá trị quan trọng. Đầu tiên, nâng cao hiệu quả và độ chính xác của mô hình khi huấn luyện với dữ liệu có nhãn nhiễu sẽ đảm bảo rằng các mô hình dự đoán, các hệ thống AI hoạt động chính xác và đáng tin cậy hơn, đặc biệt trong các lĩnh vực nhạy cảm như y tế và tài chính. Thứ hai, khả năng huấn luyện mô

hình học máy tốt trên dữ liệu nhiễu giúp tiết kiệm chi phí và nguồn lực dành cho việc thu thập và làm sạch dữ liệu, vốn là một công việc tốn kém và mất thời gian. Cuối cùng, điều này sẽ thúc đẩy sự phát triển của các ứng dụng AI mới, mở rộng khả năng ứng dụng của học máy trong các lĩnh vực mới, nơi mà dữ liệu nhiễu là một vấn đề lớn.

Như vậy, giải quyết bài toán học có giám sát với dữ liệu nhiễu không chỉ là một nhiệm vụ cần thiết mà còn mang lại nhiều giá trị quan trọng, thúc đẩy sự tiến bộ và ứng dụng của học máy trong thực tế.

1.3 Các phương pháp giải quyết bài toán

Bài toán học có giám sát với dữ liệu nhiễu là một bài toán rất được cộng đồng nghiên cứu quan tâm. Để giải quyết bài toán này có rất nhiều phương pháp đã được đưa ra và áp dụng. Có rất nhiều nghiên cứu khác nhau theo các hướng đi khác nhau với mục đích giải quyết bài toán này.

Nói đến cách để ngăn mô hình bị overfitting không thể không nói đến phương pháp Weight Decay[8]. Đây là một kỹ thuật regularization có thể giúp ngăn chặn mô hình học theo các nhãn nhiễu bằng cách thêm các ràng buộc vào quá trình huấn luyện. Tuy nhiên, Weight Decay không phải lúc nào cũng hiệu quả đặc biệt là khi dữ liệu chứa nhiều nhiễu. Khi tỷ lệ nhiễu cao, mô hình học có thể học theo các thông tin sai lệch từ dữ liệu.

Phương pháp lựa chọn dữ liệu cũng là một phương pháp đạt hiệu quả rất tốt khi có rất nhiều nghiên cứu tiêu biểu. Hướng đi của phương pháp này là giúp mô hình nhận ra được đâu là những mẫu dữ liệu tốt để học. Phương pháp lựa chọn dữ liệu có thể được thực hiện theo hai cách: *chọn mẫu* hoặc *điều chỉnh trọng số mẫu*. *Chọn mẫu* nghĩa là chọn ra một tập hợp con từ tập dữ liệu gốc sao cho tập con đó được xem là "sạch" nhất. Trong khi đó, *điều chỉnh trọng số mẫu* là đánh trọng số thấp cho những mẫu được xem là nhiễu và đánh trọng số cao cho những mẫu tốt.

Một nghiên cứu tiêu biểu đó là phương pháp Co-teaching (Han et al., 2018)[3]. Phương pháp này huấn luyện đồng thời hai mạng nơ-ron và cho phép chúng dạy lẫn nhau trên từng mini-batch dữ liệu. Với mỗi batch, mỗi mạng sẽ xem xét những trường hợp có độ mất mát nhỏ làm kiến thức hữu ích (dữ liệu sạch) để dạy cho mạng kia, giảm thiểu ảnh hưởng của dữ liệu

nhiễu. Co-teaching đã chứng minh được hiệu quả vượt trội trong việc xử lý dữ liệu nhiễu, đặc biệt là trong trường hợp dữ liệu có tỷ lệ nhiễu cao.

Sau nghiên cứu về Co-teaching, có một nghiên cứu khác đã cải tiến phương pháp này. Đó là phương pháp INCV (Chen et al., 2019)[2]. Phương pháp này sử dụng kỹ thuật cross-validation để loại bỏ dữ liệu nhiễu và xác định những dữ liệu có khả năng là tốt. Phương pháp này kết hợp với Co-teaching để sử dụng dữ liệu tốt nhất.

Một nghiên cứu khác nữa cũng đi theo một cách làm rất mới lạ nhưng mang lại hiệu quả rất tốt đó là phương pháp CRUST (Mirzasoleiman et al., 2020)[10]. Kết quả thí nghiệm của phương pháp này cũng cho thấy rằng nó đạt hiệu quả tốt hơn INCV. Phương pháp này lựa chọn tập con từ dữ liệu gốc, giữ lại các thông tin quan trọng mà không bị ảnh hưởng quá nhiều bởi nhiễu. CRUST dựa trên một số quan sát về sự phân bố của dữ liệu trên không gian gradient để tìm ra dữ liệu có nhãn sạch. Trong khóa luận này, chúng em sẽ tập trung tìm hiểu và cài đặt phương pháp CRUST sau đó đưa ra một số cải tiến nếu có thể.

Phần còn lại của khóa luận được trình bày như sau:

- Chương 2 trình bày kiến thức nền tảng về mô hình mạng nơ-ron đơn giản, các phương pháp chống quá khớp cơ bản Weight Decay, Validation và phương pháp huấn luyện với dữ liệu nhiễu INCV.

- Chương 3 trình bày về huấn luyện mô hình mạng nơ-ron với phương pháp lựa chọn dữ liệu CRUST. Trong phần này có ba phần nhỏ:

  - Dạng mô hình: trình bày mô hình mạng nơ-ron phức tạp hơn so với chương 2 mà được sử dụng trong thí nghiệm ở chương 4.
  - Huấn luyện mạng nơ-ron bằng phương pháp CRUST: trình bày cách CRUST chọn ra tập con có tỷ lệ dữ liệu sạch cao.
  - Cải tiến phương pháp CRUST: chúng em đưa ra một đề xuất cải tiến làm tăng tỷ lệ dữ liệu sạch của tập con được chọn.

- Chương 4 trình bày và phân tích về kết quả của các thí nghiệm.

- Chương 5 trình bày về các kết luận và hướng phát triển.

Chương 2

Kiến thức nền tảng

Chương này trình bày các kiến thức nền tảng của khóa luận. Đầu tiên, chúng em trình bày mô hình mạng nơ-ron đơn giản, đây là mô hình nền tảng cho mô hình chính mà khóa luận tập trung tìm hiểu (được trình bày ở chương 3). Kế tiếp, chúng em trình bày về Weight Decay và Validation - hai phương pháp cơ bản giúp chống quá khớp vào dữ liệu nhiễu trong quá trình huấn luyện mô hình. Hai phương pháp cơ bản này sẽ được dùng một cách mặc định khi huấn luyện mô hình trong các thí nghiệm (được trình bày ở chương 4). Ngoài ra, chúng em cũng trình bày về INCV - một phương pháp lựa chọn dữ liệu để giúp huấn luyện mô hình tốt hơn với dữ liệu nhiễu. Mô hình với phương pháp INCV sẽ được dùng trong phần thí nghiệm (ở chương 4) để so sánh với mô hình chính mà khóa luận tập trung tìm hiểu.

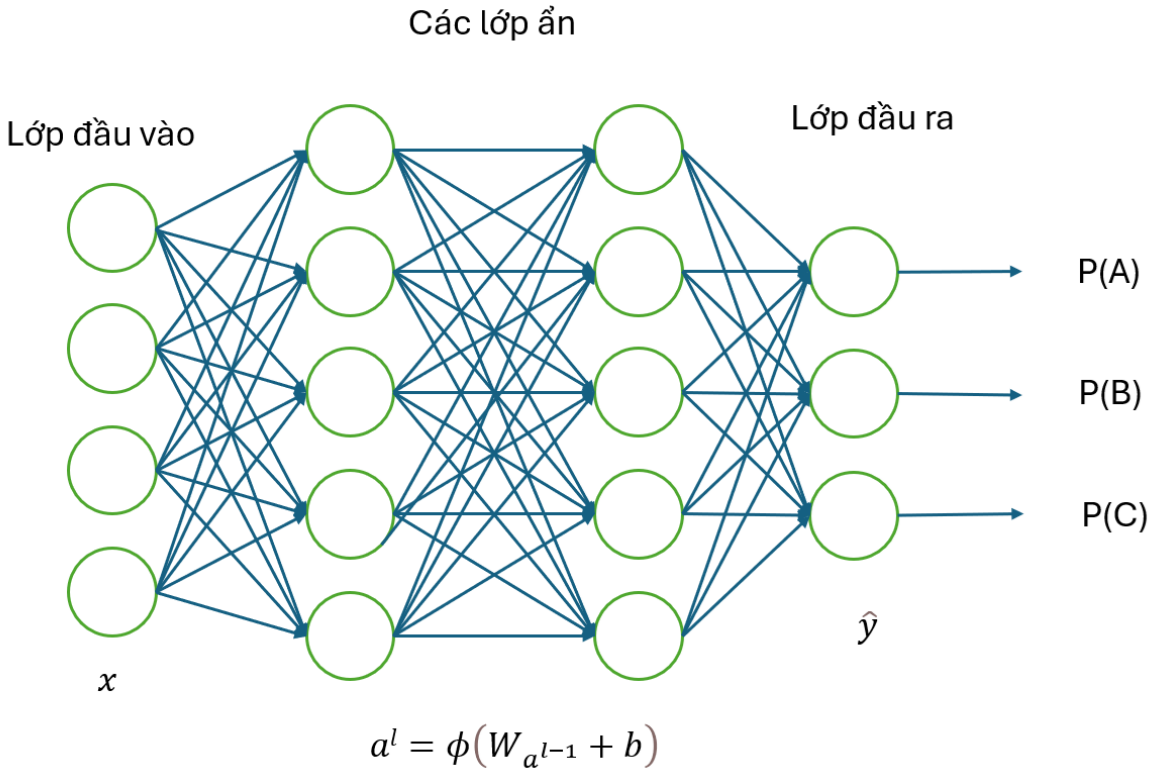
2.1 Mô hình mạng nơ-ron đơn giản

2.1.1 Dạng mô hình

Mạng nơ-ron (Neural Network - NN) là một mô hình tính toán được lấy cảm hứng từ tế bào thần kinh của con người. Đây là mô hình học rất hiệu quả, đạt kết quả cao trong các tác vụ như xử lý ngôn ngữ tự nhiên và nhận diện hình ảnh. Mô hình mạng nơ-ron có nhiều dạng/kiến trúc khác nhau, trong đó tiêu biểu là mạng lan truyền tiến kết nối đầy đủ FCFFNN (Fully-Connected FeedForward Neural Network), mạng tích chập CNN (Convolutional Neural Network), mạng hồi quy RNN (Recurrent Neural Network). Mạng FCFFNN là dạng/kiến trúc đơn giản nhất và có thể áp dụng cho dữ liệu nói chung, mạng CNN được thiết kế chuyên biệt để áp dụng cho dữ liệu hình ảnh, mạng RNN được thiết kế chuyên biệt để

áp dụng cho dữ liệu văn bản. Trong mục này, chúng em sẽ trình bày cụ thể về dạng/kiến trúc đơn giản nhất là mạng FCFFNN.

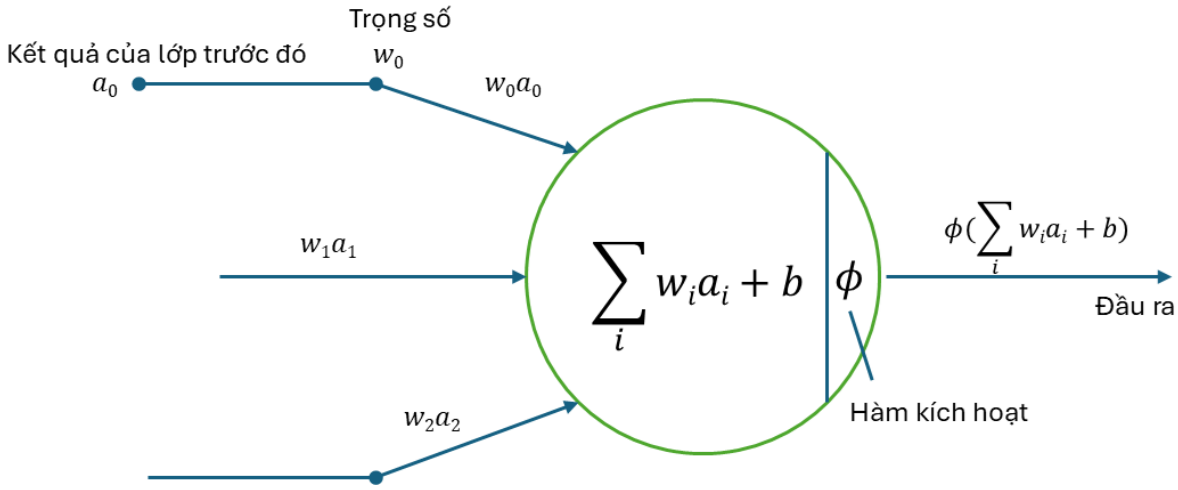
Fully-Connected FeedForward Neural Network (FCFFNN) là một loại mạng nơ-ron nhân tạo (Artificial Neural Network - ANN) trong đó các nơ-ron ở mỗi lớp (layer) đều được kết nối với tất cả các nơ-ron ở lớp tiếp theo. Điều này có nghĩa là đầu ra của mỗi nơ-ron ở lớp trước trở thành đầu vào cho mỗi nơ-ron ở lớp kế tiếp. Các kết nối này được đại diện bởi các trọng số (weights), và mỗi trọng số tương ứng với mức độ ảnh hưởng của một nơ-ron đối với nơ-ron ở lớp tiếp theo. Trong mạng FCFFNN, dữ liệu di chuyển theo một hướng duy nhất từ lớp đầu vào (input layer), qua các lớp ẩn (hidden layers), đến lớp đầu ra (output layer) và không có các kết nối quay trở lại từ lớp sau về lớp trước. Hình 2.1 trình bày FCFFNN cho bài toán phân lớp. Trong khóa luận này, chúng em cũng chỉ tập trung trình bày kiến trúc FCFFNN cho bài toán phân lớp - một trường hợp của bài toán học có giám sát.



Hình 2.1: Fully-Connected FeedForward Neural Network với 2 lớp ẩn cho bài toán phân lớp.

Các thành phần cơ bản của mạng nơ-ron

Trước khi đi vào tìm hiểu về mạng nơ-ron và FCFFNN, chúng em sẽ giải thích khái niệm về các thành phần mà mạng nơ-ron nào cũng có, các khái niệm này sẽ sử dụng để trình bày xuyên suốt chương này và cả các chương sau. Khi nói về mạng nơ-ron, chúng ta cần hiểu các khái niệm sau: nơ-ron, weight (trọng số), bias (độ lệch), activation function (hàm kích hoạt). Hình 2.2 trình bày mối quan hệ giữa các thành phần trên.



Hình 2.2: Hình minh họa cho một nơ-ron trong học máy.

Chức năng cụ thể của các thành phần trên là:

- **Nơ-ron**: là thành phần cơ bản của mạng nơ-ron. Nhận tín hiệu là các kết quả đầu ra của các nơ-ron trước đó, sau đó xử lý bằng phép tổng gia trọng các tín hiệu đầu vào rồi áp dụng một hàm kích hoạt để xuất ra giá trị duy nhất. Giá trị này là tín hiệu để truyền qua nơ-ron tiếp theo.

- **Trọng số (weight)**: thường được ký hiệu là **w**. Là các giá trị điều chỉnh sức ảnh hưởng của đầu vào đến nơ-ron. Mỗi liên kết giữa các nơ-ron được gán một trọng số, và các trọng số này được điều chỉnh trong quá trình huấn luyện để giảm sai số đầu ra của mạng. Trọng số quyết định tầm quan trọng của các tín hiệu đầu vào.

- **Độ lệch (bias)**: thường được ký hiệu là **b**. Là giá trị được thêm vào phép nhân trọng số trước khi đưa vào hàm kích hoạt. Giá trị

này giúp dịch chuyển kết quả đầu ra của phép tính nhân trọng số, vì phép tính này luôn là đường thẳng đi qua gốc tọa độ. Việc làm này sẽ giúp mô hình linh hoạt hơn, có khả năng khớp tốt hơn với dữ liệu.

- **Hàm kích hoạt (activation function)**: là một thành phần quan trọng của mạng nơ-ron. Hàm kích hoạt có thể giúp mô hình học các mối quan hệ phi tuyến tính. Hàm kích hoạt còn có khả năng chuẩn hóa đầu ra của các nơ-ron, ví dụ như hàm sigmoid chuẩn hóa đầu ra trong khoảng (0,1). Các hàm kích hoạt thường được sử dụng là:
  - **Hàm Sigmoid**: chuyển đổi giá trị đầu ra thành một giá trị trong khoảng từ 0 đến 1.
  - **Hàm Tanh (Hyperbolic Tangent)**: chuyển đổi giá trị đầu ra thành một giá trị trong khoảng từ -1 đến 1.
  - **Hàm ReLU (Rectified Linear Unit)**: chuyển đổi giá trị đầu ra thành 0 nếu đầu vào nhỏ hơn 0, và giữ nguyên giá trị đầu vào nếu đầu vào lớn hơn hoặc bằng 0.

Kiến trúc của Fully-Connected FeedForward Neural Network (FCFFNN)

FCFFNN gồm có ba phần chính đó là lớp đầu vào (input layer), các lớp ẩn (hidden layers) và lớp đầu ra (output layer).

- **Lớp đầu vào (Input Layer)**: chức năng chính của lớp này là tiếp nhận dữ liệu từ môi trường bên ngoài vào mạng nơ-ron. Dữ liệu nhận vào được biểu diễn dưới dạng các vector đặc trưng. Giả sử mỗi mẫu trong dữ liệu có d đặc trưng thì vector đặc trưng là $\mathbf{x}_i \in \mathbb{R}^d$ hay $\mathbf{x}_i = \left[x_{i_1}, x_{i_2}, \ldots, x_{i_d}\right]^T$. Lớp này chỉ thực hiện truyền dữ liệu, không có phép tính nào được diễn ra trong lớp này.
- **Các lớp ẩn (Hidden Layers)**: thực hiện các phép tính toán, biến đổi phức tạp để giúp mạng nơ-ron học các mối quan hệ, thông tin quan trọng giữa giữ liệu đầu vào và đầu ra. Với kiến trúc FCFFNN thì mỗi nơ-ron trong mỗi lớp ẩn nhận đầu vào từ tất cả các nơ-ron của lớp trước đó sau đó áp dụng một hàm kích hoạt lên tổng có trọng

số của các đầu vào. Có thể có một hoặc nhiều lớp ẩn, kích thước của lớp ẩn do người tạo ra mô hình quyết định. Số lượng lớp ẩn, số nơ-ron trong mỗi lớp ẩn sẽ ảnh hưởng trực tiếp đến khả năng học và hiệu suất của mô hình nên chúng thường dựa vào kích thước và độ phức tạp của dữ liệu.

- **Lớp đầu ra (Output Layer)**: lớp đầu ra nhận kết quả từ lớp ẩn cuối cùng và chuyển đổi thành đầu ra của mạng (kết quả dự đoán). Số lượng nơ-ron trong lớp này tương ứng với số lớp đầu ra mong muốn của bài toán. Ví dụ trong bài toán phân lớp chữ số viết tay thì có 10 nơ-ron cho 10 lớp, với bài toán phân loại nhị phân thì chỉ cần 1 nơ-ron. Lớp này thường áp dụng các hàm kích hoạt như softmax (phân loại nhiều lớp) hoặc sigmoid (phân loại nhị phân) để đưa ra xác suất của các lớp.

Quá trình huấn luyện của Fully-Connected FeedForward Neural Network (FCFFNN)

Quá trình huấn luyện của một Fully-Connected FeedForward Neural Network (FCFFNN) bao gồm hai giai đoạn: giai đoạn FeedForward và giai đoạn backpropagation.

Giai đoạn FeedForward: trong giai đoạn này, dữ liệu đưa vào mạng sẽ được lan truyền tới phía trước của mạng. Tại mỗi lớp ẩn, các nơ-ron nhận đầu vào là tất cả đầu ra của nơ-ron trong lớp trước đó, mỗi đầu ra được nhân với một trọng số tương ứng, sau cùng được cộng thêm một hệ số độ lệch. Kết quả sau đó được đưa qua hàm kích hoạt, rồi tiếp tục truyền đến nơ-ron tiếp theo. Quá trình này thực hiện liên tục cho đến lớp cuối cùng và xuất ra kết quả dự đoán. Hình 2.1 minh họa quá trình lan truyền tiến trên mạng và hình 2.2 minh họa cho các phép tính thực hiện ở mỗi nơ-ron. Tóm lại, các nơ-ron tổng hợp tổng thông tin tại lớp ẩn l theo công thức sau:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

Kết quả sau cùng sẽ được đưa qua hàm kích hoạt ϕ:

$$\mathbf{a}^{(l)} = \phi(\mathbf{z}^{(l)})$$

Trong đó:

- $\mathbf{z}^{(l)}$: vector đầu ra chưa qua hàm kích hoạt của lớp l.
- $\mathbf{W}^{(l)}$: ma trận trọng số có kích thước $m \times n$, với m là số nút trong lớp l và n là số nút trong lớp $(l-1)$.
- $\mathbf{a}^{(l-1)}$: vector đầu ra của lớp $(l-1)$.
- $\mathbf{b}^{(l)}$: vector bias của lớp l, có kích thước m.

Giai đoạn backpropagation: Sau khi thực hiện dự đoán, mô hình sẽ sử dụng hàm lỗi huấn luyện so sánh sự khác nhau giữa đầu ra và kết quả thực tế. Mỗi hàm lỗi huấn luyện khác nhau sẽ có cách so sánh khác nhau. Độ lỗi này sẽ được lan truyền ngược qua mạng, sau đó mô hình sẽ phải điều chỉnh các trọng số để giảm thiểu độ lỗi này. Thông thường, người ta sử dụng thuật toán Gradient Descent hoặc các biến thể của nó như Batch Gradient Descent, Stochastic Gradient Descent, Mini-batch Gradient Descent,... để tối ưu độ lỗi huấn luyện.

2.1.2 Huấn luyện mô hình bằng Gradient Descent

Trong giai đoạn backpropagation, mô hình thực hiện cập nhật trọng số để tối ưu hàm lỗi huấn luyện. Trong mục này, chúng em sẽ trình bày về thuật toán Gradient Descent, một thuật toán phổ biến để làm điều này. Mặt khác, vì trong khóa luận này chúng em chỉ tập trung vào bài toán phân lớp, hàm lỗi huấn luyện được sử dụng sẽ là Cross Entropy.

Hàm lỗi huấn luyện Cross Entropy

Trong bài toán phân lớp của học máy, hàm lỗi Cross Entropy được sử dụng rất phổ biến để đánh giá mô hình. Hàm lỗi Cross Entropy đo lường sự khác biệt giữa hai phân phối xác suất: phân phối xác suất thực tế (các nhãn thật) và phân phối xác suất dự đoán (các nhãn mà mô hình dự đoán).

Mục tiêu của mô hình là làm cho phân phối xác suất mà nó dự đoán ra càng gần với phân phối xác suất thực tế càng tốt. Cross Entropy cung cấp một thước đo định lượng để đánh giá khoảng cách này.

Đối với bài toán phân lớp nhị phân, giả sử nhãn dự đoán của chúng ta là $\hat{y}$ và nhãn thực tế là y, trong đó $y \in \{0, 1\}$. Công thức tính độ lỗi Cross Entropy cho một mẫu duy nhất là:

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

Trong trường hợp phân lớp có nhiều lớp khác nhau, giả sử cụ thể có C lớp, công thức tính độ lỗi Cross Entropy cho một mẫu duy nhất là:

$$L(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Khi huấn luyện, mô hình sẽ được học với rất nhiều mẫu khác nhau. Tổng độ lỗi trên toàn bộ dữ liệu huấn luyện được tính bằng cách lấy trung bình của độ lỗi trên từng mẫu dữ liệu:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(y_i, \hat{y}_i)$$

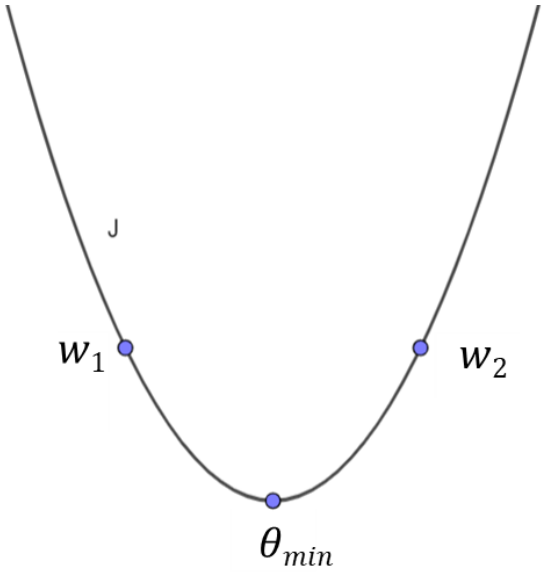
Trong đó:

- $J(\theta)$ là hàm lỗi tổng hợp.
- m là số lượng mẫu dữ liệu trong tập huấn luyện.
- θ là các tham số của mô hình.

Thuật toán Gradient Descent

Gradient Descent là một thuật toán tối ưu hóa hàm lỗi huấn luyện được sử dụng phổ biến trong học máy. Mục tiêu của huấn luyện là cố gắng làm cho hàm lỗi huấn luyện đạt giá trị nhỏ nhất có thể. Bằng cách cập nhật các tham số mô hình theo hướng ngược chiều với gradient của hàm lỗi huấn luyện, thuật toán đã làm được điều nêu trên. Để hiểu rõ hơn tại

sao phương pháp này hoạt động, chúng ta sẽ xem xét ví dụ cụ thể được trình bày dưới đây (xem minh họa ở hình 2.3):



Hình 2.3: Hình minh họa hàm lỗi huấn luyện J có dạng parabol.

- Giả sử hàm lỗi huấn luyện $J(\theta)$ là một hàm lồi, có dạng đơn giản như một đường parabol. Điểm cực tiểu của hàm này là $\theta = \theta_{\min}$. Chúng ta cần tìm giá trị θ sao cho $J(\theta)$ đạt giá trị nhỏ nhất.

- Gradient của hàm lỗi $J(\theta)$ tại $\theta = w_i$ được ký hiệu là $\nabla J(w_i)$ và được định nghĩa là đạo hàm của J tại w_i:

$$\nabla J(w_i) = \frac{d}{d\theta} J(\theta)\Big|_{\theta=w_i}$$

- Cập nhật tham số theo Gradient Descent được thực hiện bằng cách di chuyển tham số w_i ngược chiều với gradient của hàm lỗi. Như vậy, cập nhật này được thực hiện theo công thức:

$$w_{i+1} = w_i - \eta \nabla J(w_i)$$

  Trong đó:

  – η là kích thước bước (learning rate).

– $\nabla J(w_i)$ là gradient của hàm lỗi tại w_i.

- Trong trường hợp w_i lớn hơn giá trị cực tiểu ($w_i = w_2 > \theta_{\min}$) thì:
  - Đạo hàm $\nabla J(w_i)$ sẽ dương, bởi vì hàm lỗi đang đồng biến (khi θ tăng thì $J(\theta)$ cũng tăng).
  - Cập nhật tham số sẽ là:

$$w_{i+1} = w_i - \eta \nabla J(w_i)$$

    Vì $\nabla J(w_i)$ dương, tham số w_i sẽ giảm. Điều này giúp di chuyển w_i về phía điểm cực tiểu $\theta_{\min}$.

- Trong trường hợp w_i nhỏ hơn giá trị cực tiểu ($w_i = w_1 < \theta_{\min}$) thì:
  - Đạo hàm $\nabla J(w_i)$ sẽ âm, bởi vì hàm lỗi đang nghịch biến (khi θ tăng thì $J(\theta)$ giảm).
  - Cập nhật tham số sẽ là:

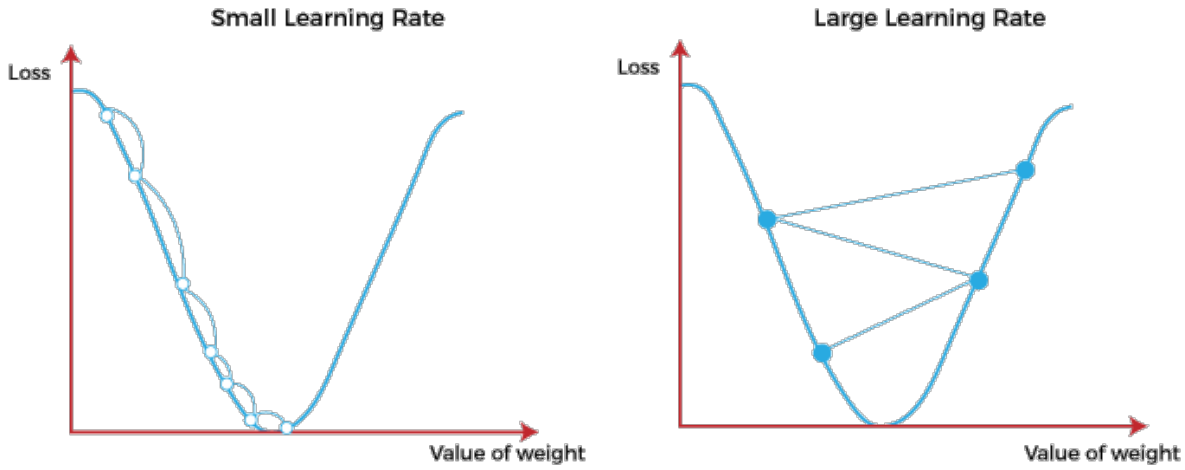
$$w_{i+1} = w_i - \eta \nabla J(w_i)$$

    Vì $\nabla J(w_i)$ âm, tham số w_i sẽ tăng. Điều này giúp di chuyển w_i về phía điểm cực tiểu $\theta_{\min}$.

- Như vậy, cho dù tham số w_i là giá trị nào đi nữa thì việc cập nhật tham số theo hướng ngược chiều gradient đều làm cho độ lỗi tiến đến giá trị cực tiểu, làm giảm độ lỗi huấn luyện.

Thuật toán Gradient Descent là một thuật toán tốt để tối ưu hàm lỗi huấn luyện. Có một lưu ý khi sử dụng khi sử dụng thuật toán này đó chính là chọn learning rate η phù hợp sẽ giúp hàm lỗi huấn luyện đạt đến cực tiểu nhanh hơn và mô hình sẽ có độ chính xác cao hơn. Nếu chọn learning rate η nhỏ, sẽ cần phải tốn rất nhiều cập nhật thì w_i mới đạt đến cực tiểu $\theta_{\min}$. Ngược lại, nếu chọn learning rate η quá lớn, lúc cập nhật có thể trực tiếp bỏ qua cực tiểu, điều này có thể làm cho mô hình thậm chí không bao giờ hội tụi được hoặc phải tốn rất nhiều lần cập nhật. Hình 2.4 minh họa

việc chọn learning rate η quá lớn và quá nhỏ sẽ ảnh hưởng đến việc học như thế nào.



Hình 2.4: Learning rate thấp và cao.[1]

Tóm lại, Gradient Descent là một thuật toán tối ưu hóa sử dụng để tìm giá trị của các tham số mô hình θ sao cho hàm lỗi $J(\theta)$ được cực tiểu hóa. Thuật toán Gradient Descent bao gồm các bước sau:

1. **Khởi tạo**: Khởi tạo các tham số θ bằng các giá trị ngẫu nhiên hoặc giá trị cố định nào đó.

2. **Tính toán Gradient**: Với mỗi tham số θ, tính gradient của hàm lỗi $J(\theta)$, gradient là vector chứa đạo hàm riêng phần của hàm lỗi đối với từng tham số:

$$\nabla_\theta J(\theta) = \left[\frac{\partial J(\theta)}{\partial \theta_1}, \frac{\partial J(\theta)}{\partial \theta_2}, \ldots, \frac{\partial J(\theta)}{\partial \theta_n}\right]$$

3. **Cập nhật tham số**: Cập nhật tham số với learning rate η bằng cách cộng chúng với một lượng bằng âm cuả gradient nhân với learning rate. Cụ thể, công thức cập nhật là:

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

4. **Lặp lại**: Liên tục lặp lại thực hiện bước 2 và bước 3 cho đến khi

hàm lỗi hội tụ đến giá trị tối thiểu hoặc đạt đến số lượng vòng lặp tối đa đã định trước.

Với bài toán phân lớp đa lớp được trình bày trong khóa luận, hàm lỗi huấn luyện được sử dụng sẽ là Cross Entropy. Trong bước 3, trước khi cập nhật tham số θ, gradient của hàm lỗi này được tính theo công thức sau:

$$\nabla_\theta J = \sum_{i=1}^{N} \sum_{j=1}^{C} (\hat{y}_{ij} - y_{ij}) \mathbf{a}_i$$

Trong đó:

- $\hat{y}_{ij}$ là xác suất dự đoán của lớp thứ j cho mẫu i.
- y_{ij} là nhãn thực tế của lớp thứ j cho mẫu i.
- $\mathbf{a}_i$ là vector đầu ra của lớp nơ-ron trước đó cho mẫu i.
- N là số lượng mẫu của dữ liệu huấn luyện.
- C là số lớp của bài toán.

Tổng kết, thuật toán Gradient Descent thực hiện lặp đi lặp lại quá trình cập nhật tham số theo hướng ngược với gradient của hàm lỗi huấn luyện. Việc này sẽ giúp giảm độ lỗi của hàm lỗi huấn luyện. Bên cạnh đó, learning rate ảnh hưởng trực tiếp đến tốc độ hội tụ của thuật toán này. Ngoài ra, thuật toán Gradient Descent có nhiều biến thể khác như Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent. Hơn nữa, người ta cũng phát triển ra các phương pháp tối ưu hóa khác giúp mô hình đạt hiệu quả tốt hơn và tăng tốc độ hội tụ như: Adam, RMSprop,...

2.2 Phương pháp chống quá khớp Weight Decay và Validation

Weight Decay (Krogh và Hertz, 1992)[8], còn có tên gọi khác là Ł2 regularization, là một kỹ thuật không còn xa lạ, được sử dụng phổ biến để giảm thiểu overfitting.

[1]Nguồn: Javatpoint.com

Phương pháp thêm một số hạng nữa vào hàm lỗi huấn luyện nhằm làm giảm độ phức tạp của mô hình và từ đó ngăn chặn overfitting. Cụ thể, với kỹ thuật regularization này sẽ thêm norm bậc hai của các trọng số của mô hình. Hàm lỗi huấn luyện được điều chỉnh như sau:

$$\mathcal{L}_{reg} = \mathcal{L}_{original} + \lambda \sum_{j=1}^{n} w_j^2$$

trong đó $\mathcal{L}_{original}$ là hàm lỗi huấn luyện gốc, w_j là các trọng số của mô hình và λ là hệ số điều chỉnh mức độ regularization.

Tuy nhiên, Weight Decay có thể khó đạt được hiệu suất tốt khi dữ liệu có tỷ lệ nhãn nhiễu cao do nó không phân biệt được giữa dữ liệu sạch và dữ liệu nhiễu.

Mặc dù vậy, đây là một phương pháp rất tốt và hiệu quả trong việc ngăn chặn mô hình bị overfitting, thế nên phương pháp này thường được áp dụng cho hầu hết các mạng nơ-ron khi cài đặt. Trong phần thí nghiệm của chúng em, mô hình nền của tất cả các phương pháp lựa chọn dữ liệu đều được cài Weight Decay.

Một phương pháp khác nữa giúp ngăn chặn overfitting một cách hiệu quả đó chính là sử dụng validation set. Validation set (tập kiểm định) là tập con của dữ liệu được sử dụng trong quá trình huấn luyện nhằm theo dõi hiệu suất của mô hình, từ đó phát hiện vấn đề overffiting sớm và điều chỉnh siêu tham số hoặc các biện pháp khác trước khi thực hiện kiểm thử trên test set. Nếu ta lấy ví dụ test set là đề thi trung học phổ thông quốc gia, validation set sẽ là các đề thi thử, nhằm kiểm tra tình hình học tập như thế nào để có thể điều chỉnh và bổ sung kiến thức phù hợp. Mặc dù phương pháp này sẽ làm cho dữ liệu huấn luyện ít đi một phần, nó giúp ngăn chặn overfitting hiệu quả. Hơn nữa, vì dễ thực hiện, nó được sử dụng rộng rãi trong các thí nghiệm học máy.

2.3 Phương pháp lựa chọn dữ liệu INCV

2.3.1 Giới thiệu

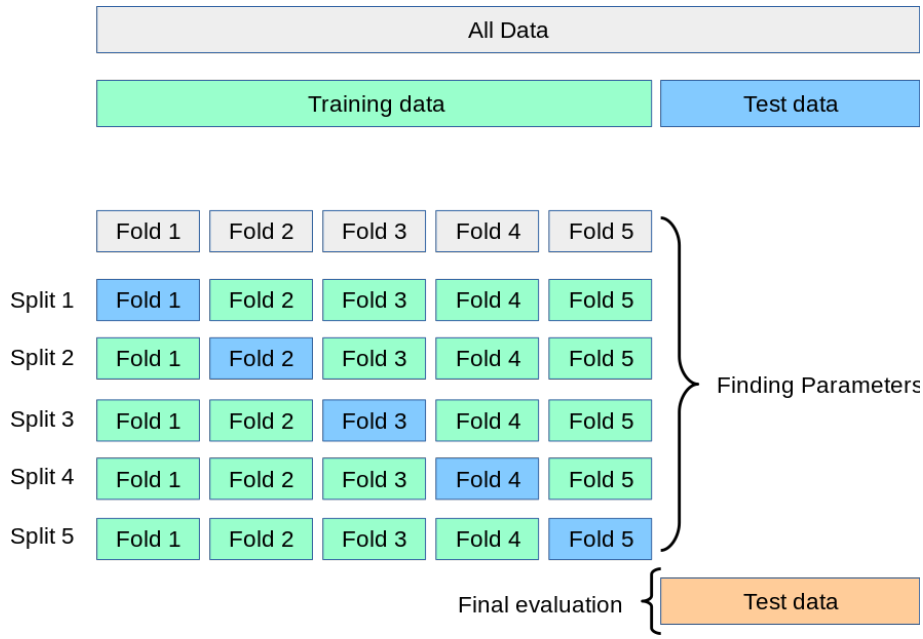
Iterative Noisy Cross-Validation (INCV)[2] được trình bày trong bài báo "Understanding and Utilizing Deep Neural Networks Trained with Noisy Labels" là một phương pháp lựa chọn dữ liệu sạch được giới thiệu bởi Chen cùng cộng sự vào năm 2019 tại hội nghị ICML. Phương pháp này dùng kỹ thuật Cross-Validation để chia dữ liệu một cách ngẫu nhiên rồi xác định những mẫu là dữ liệu sạch. Sau đó thực hiện huấn luyện dữ liệu này bằng phương pháp Co-teaching. Tại thời điểm ra mắt, phương pháp này được đánh giá rất cao so sánh với các phương pháp khác, nó đạt hiệu suất state-of-the-art.

2.3.2 Nguyên lý hoạt động INCV

Cross-Validation

Cross-Validation là một kỹ thuật trong học máy giúp đánh giá hiệu suất mô hình một cách chính xác được sử dụng rất phổ biến. Kỹ thuật này giúp chúng ta sử dụng dữ liệu hiệu quả hơn, phát hiện và giúp mô hình tránh khỏi overffiting. Kỹ thuật Cross-Validation chia dữ liệu thành nhiều phần khác nhau, khi phần dữ liệu này được sử dụng để huấn luyện, các phần khác sẽ được sử dụng để kiểm định. Quá trình này sẽ lặp lại lần lượt qua hết các phần của dữ liệu để đảm bảo kết quả đánh giá là toàn diện nhất. Các kiểu Cross-Validation phổ biến là:

- **K-Fold Cross-Validation:** chia dữ liệu thành k phần bằng nhau, lần lượt huấn luyện trên một phần và kiểm định trên k-1 phần còn lại (Xem hình 2.5).
- **Stratified K-Fold Cross-Validation:** thực hiện tương tự như K-Fold nhưng thêm điều kiện là tỷ lệ các lớp trong mỗi fold phải bằng tỷ lệ trong tập dữ liệu gốc.
- **Leave-One-Out Cross-Validation:** mỗi lần huấn luyện chỉ sử dụng một mẫu và phần còn lại của dữ liệu đều dùng để kiểm định.



Hình 2.5: K-Fold Cross-Validation. [2]

Co-teaching

Co-teaching được giới thiệu lần đầu bởi Blum và Mitchell (1998)[1]. Đến năm 2018, Han cùng cộng sự đã cải tiến để giải quyết bài toán huấn luyện mạng nơ-ron với dữ liệu nhiễu[3] và đạt được hiệu suất khá tốt. Phương pháp Co-teaching dựa trên việc huấn luyện đồng thời hai mô hình. Mỗi mô hình sẽ chọn ra một tập hợp các mẫu dữ liệu có độ lỗi nhỏ trong quá trình huấn luyện từ tập dữ liệu và chia sẻ tập hợp này với mô hình kia để huấn luyện. Phương pháp này giúp mô hình học tốt hơn và loại bỏ các mẫu nhiễu một cách hiệu quả. Tuy nhiên, khi tỷ lệ nhiễu quá cao, hiệu suất của phương pháp này sẽ giảm đáng kể. Phương pháp INCV được đề xuất để giải quyết vấn đề này. Nó sẽ chọn ra một tập dữ liệu có tỷ lệ nhiễu thấp hơn nhiều so với dữ liệu gốc, sau đó sử dụng phương pháp Co-teaching để huấn luyện.

2.3.3 Chi tiết thuật toán

Dữ liệu sạch trong phương pháp INCV được xác định bằng cách huấn luyện một hàm dự đoán $f(x, \omega)$ đủ tốt (x là vector đặc trưng, ω là tham số mô hình), sau đó thực hiện dự đoán trên điểm dữ liệu (x, y). Nếu kết quả

[2]Nguồn: scikit-learn.org

dự đoán là y^f giống với nhãn của nó là y, thì điểm dữ liệu này được xem là dữ liệu sạch. Để thực hiện dự đoán như vậy, tất nhiên điểm dữ liệu (x, y) sẽ không được sử dụng để huấn luyện mô hình vì có khả năng mô hình bị overffiting. Vì vậy, phương pháp INCV sử dụng kỹ thuật Cross-Validation để huấn luyện trên một phần của dữ liệu và dự đoán trên phần còn lại. Ban đầu, tác giả đề xuất lựa chọn dữ liệu sạch bằng thuật toán sau[2]:

Algorithm 1 Lựa chọn dữ liệu sạch bằng Noisy Cross-Validation (NCV)

Đầu vào: Tập dữ liệu nhiễu D, số epoch E

1: Khởi tạo $S = \emptyset$ và mô hình $f(x, \omega)$
2: Chia dữ liệu ngẫu nhiên thành hai tập D_1 và D_2
3: Huấn luyện mô hình f trên D_1 trong E epoch
4: Chọn những mẫu sạch trên D_2 có nhãn dự đoán bằng nhãn của chính nó: $S_1 = \{(x, y) \in D_2 \mid y^f = y\}$
5: Khởi tạo lại mô hình $f(x, \omega)$
6: Huấn luyện mô hình f trên D_2 trong E epoch
7: Chọn những mẫu sạch trên D_1 có nhãn dự đoán bằng nhãn của chính nó: $S_2 = \{(x, y) \in D_1 \mid y^f = y\}$
8: Tập dữ liệu sạch S là $S = S_1 \cup S_2$

Đầu ra: Tập dữ liệu sạch S

Có thể thấy để thuật toán trên đưa ra kết quả tốt, kết quả dự đoán từ hàm f phải cao hơn tỷ lệ dữ liệu sạch. Thực tế, các kết quả thí nghiệm đều chỉ ra rằng nhận định này là khả thi. Tuy nhiên, nó chỉ đúng khi tỷ lệ nhiễu khá thấp và trong những trường hợp đó thì độ chính xác của mô hình cũng không cao hơn tỷ lệ dữ liệu sạch quá nhiều. Để nâng cao hiệu quả của quá trình lựa chọn dữ liệu sạch, tác giả cải tiến cách làm trên bằng việc lặp đi lặp lại nhiều lần Cross-Validation. Thêm vào đó, thực hiện loại bỏ các mẫu có độ lỗi huấn luyện cao vì các mẫu này khả năng cao là dữ liệu nhiễu. Cụ thể, cách làm được trình bày trong thuật toán sau[2]:

Algorithm 2 Lựa chọn dữ liệu sạch bằng Iterative Noisy Cross-Validation (INCV)

Đầu vào: Tập dữ liệu nhiễu D, số lần lặp N, số epoch E, tỷ lệ loại bỏ r

1: Tập dữ liệu được chọn $S = \emptyset$, tập dữ liệu ứng viên $C = D$
2: **for** $i = 1, \ldots, N$ **do**
3: Khởi tạo mạng $f(x; \omega)$
4: Chia ngẫu nhiên C thành hai phần C_1 và C_2
5: Huấn luyện mô hình f trên $S \cup C_1$ trong E epoch
6: Chọn các mẫu sạch trên C_2 có nhãn dự đoán bằng nhãn của chính nó: $S_1 = \{(x, y) \in C_2 \mid y^f = y\}$
7: Loại bỏ những mẫu có độ lỗi huấn luyện lớn nhất: Xác định $n = r|S_1|$ và loại bỏ $R_1 = \{\#n \arg\max_{C_2} L(y, f(x; \omega))\}$
8: Khởi tạo lại mạng $f(x; \omega)$
9: Huấn luyện mô hình f trên $S \cup C_2$ trong E epoch
10: Chọn các mẫu sạch trên C_1 có nhãn dự đoán bằng nhãn của chính nó: $S_2 = \{(x, y) \in C_1 \mid y^f = y\}$
11: Loại bỏ những mẫu có độ lỗi huấn luyện lớn nhất: Xác định $n = r|S_2|$ và loại bỏ $R_2 = \{\#n \arg\max_{C_1} L(y, f(x; \omega))\}$
12: Cập nhật tập dữ liệu sạch và tập ứng viên: $S = S \cup S_1 \cup S_2$, $C = C - (S_1 \cup S_2 \cup R_1 \cup R_2)$
13: **end for**

Đầu ra: Tập dữ liệu sạch S

Sau khi chọn ra tập dữ liệu sạch, nó có thể được huấn luyện bằng bất kỳ phương pháp nào. Trong bài báo gốc, tác giả thực hiện bằng Co-teaching, một phương pháp huấn luyện khá tốt với dữ liệu nhiễu tại thời điểm đó. Kết quả thí nghiệm của Co-teaching sau khi thực hiện chọn dữ liệu sạch bằng INCV đã cải thiện đáng kể, đặc biệt là với những trường hợp nhiễu nặng. Tóm lại, đây là một phương pháp tốt để huấn luyện mô hình với dữ liệu nhiễu.

Chương 3

Mô hình mạng nơ-ron với phương pháp lựa chọn dữ liệu CRUST

Chương này trình bày mô hình mạng nơ-ron với phương pháp lựa chọn dữ liệu CRUST[10] được đề xuất để giải quyết bài toán học có giám sát với dữ liệu nhiễu; đây là mô hình chính mà chúng em tập trung tìm hiểu trong khóa luận. Ở phần đầu tiên, chúng em trình bày dạng của mô hình mạng nơ-ron; ở chương 2 chúng em đã trình bày một kiến trúc mạng đơn giản, ở đây chúng em trình bày một kiến trúc mạng phức tạp hơn mà sẽ được sử dụng trong phần thí nghiệm ở chương 4. Ở phần thứ hai, chúng em trình bày cách huấn luyện mạng nơ-ron bằng phương pháp lựa chọn dữ liệu CRUST; với phương pháp này, ở mỗi vòng lặp của Gradient Descent đã trình bày ở chương 2, ta không dùng toàn bộ dữ liệu để tính gradient và cập nhật trọng số, mà chỉ dùng một tập con được lựa chọn sao cho đa số trong đó là dữ liệu sạch. Cuối cùng, chúng em đưa ra một đề xuất cải tiến đơn giản cho phương pháp CRUST để có thể tăng tỉ lệ dữ liệu sạch trong tập con được chọn.

3.1 Dạng mô hình

Phương pháp CRUST là một phương pháp giúp tìm ra tập con dữ liệu sạch có thể đại diện cho toàn bộ dữ liệu từ dữ liệu nhiễu, sau đó sử dụng mạng nơ-ron huấn luyện trên tập dữ liệu vừa tìm ra đó. Phương pháp này có thể áp dụng cho một mạng nơ-ron có dạng/kiến trúc bất kỳ như mạng lan truyền tiến kết nối đầy đủ FCFFNN (Fully-Connected FeedForward Neural Network), mạng tích chập CNN (Convolutional Neural Network), mạng hồi quy RNN (Recurrent Neural Network). Ở chương 2, chúng em đã trình bày cụ thể về dạng/kiến trúc đơn giản nhất là mạng FCFFNN.

Trong mục này, chúng em sẽ trình bày cụ thể về dạng/kiến trúc của mạng CNN cho dữ liệu ảnh; chúng em sẽ dùng mạng CNN trong phần thí nghiệm ở chương 4.

3.1.1 Giới thiệu

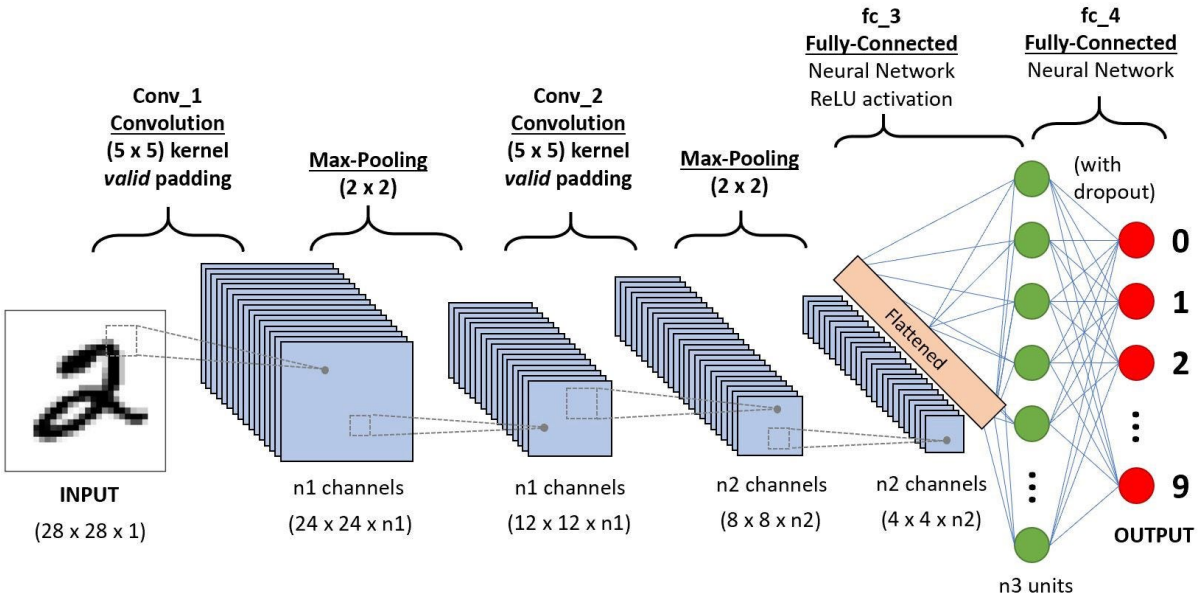
Mạng CNN (Convolutional Neural Network) là một dạng mô hình/kiến trúc được thiết kế đặc biệt để xử lý dữ liệu dạng hai chiều, có ứng dụng rộng rãi trong các bài toán thị giác máy tính như phân loại hình ảnh, phát hiện vật thể, nhận diện khuôn mặt,... Từ "convolution" trong tên của dạng mô hình này có nghĩa là tích chập, đây chính là phép toán cốt lõi để tạo nên mạng CNN. Trong bài toán xử lý ảnh, các phép tích chập giữa ma trận ảnh và ma trận bộ lọc giúp mô hình trích xuất các đặc trưng của bức ảnh một cách chính xác, từ đó giúp mô hình ghi nhớ và học được các thông tin hữu ích của bức ảnh hiệu quả hơn.

Mạng CNN được đề xuất lần đầu vào năm 1988 bởi Yann Lecun[9] nhằm cải tiến vấn đề về nhận dạng chữ cái viết tay. Tại thời điểm đó kiến trúc mạng nơ-ron chủ yếu được sử dụng là FCFFNN (Fully-Connected FeedForward Neural Network), mô hình này không giữ được cấu trúc không gian của dữ liệu nên khó khăn trong việc học mối quan hệ giữa các pixel ảnh liền kề. Hơn nữa, chi phí để huấn luyện nó cũng rất cao. Giả sử, đầu vào là môt bức ảnh RGB với kích thước 256x256 và lớp ẩn đầu tiên của mô hình FCFFNN có 500 nơ-ron, sẽ phải cần 256x256x3x500 $\sim$ 98 triệu tham số chỉ riêng cho lớp ẩn đầu tiên. Nếu làm theo kiến trúc của CNN, việc học không chỉ hiệu quả hơn mà còn nhanh hơn nhiều. Đó chính là lý do vì sao dạng mô hình này được sử dụng nhiều trong bài toán xử lý ảnh và cũng là lý do chúng em sử dụng để thực hiện thí nghiệm bài toán phân loại ảnh với CRUST. Phần dưới đây chúng em sẽ trình bày cụ thể hơn về CNN.

3.1.2 Các thành phần cơ bản của mạng CNN

Kiến trúc tổng thể

Mạng CNN bao gồm ba loại lớp ẩn xếp chồng lên nhau, bao gồm lớp tích chập - convolution layer, lớp gộp - pooling layer và lớp kết nối đầy đủ - fully-connected layer. Hình 3.1 là hình minh họa kiến trúc CNN đơn giản cho bài toán phân loại chữ số viết tay với tập MNIST.



Hình 3.1: Mạng CNN đơn giản cho bài toán nhận dạng chữ số viết tay trên tập dữ liệu MNIST.[2]

Có thể tóm tắt chức năng của các thành phần chính có mặt trong mạng CNN như sau:

- Lớp input, cũng giống như các dạng ANN khác, sẽ chứa đầu vào các pixel của bức ảnh, nhưng sẽ giữ ở cấu trúc 2 chiều.
- Lớp tích chập - convolution layer, thực hiện các phép tích chập giữa các vùng cục bộ của dữ liệu và bộ lọc (filter) để trích xuất các đặc trưng của hình ảnh; ví dụ như cạnh, góc hoặc các hình dạng phức tạp khác.

[2]Nguồn: medium.com

- Lớp gộp - pooling layer, có chức năng chính là giảm kích thước không gian đặc trưng, từ đó giảm số tham số và lượng tính toán trong mạng, giúp mạng trở nên đơn giản hơn.

- Lớp kết nối đầy đủ - fully-connected layer có chức năng chính là kết hợp các đặc trưng đã trích xuất từ các lớp trước đó và tạo ra các dự đoán.

Mạng CNN là loại mạng nơ-ron sử dụng các loại lớp trên, sắp xếp chúng chồng lên nhau một một thứ tự hợp lý. Ngoài ra, trong một số kiến trúc cụ thể như ResNet còn có thể áp dụng một số kỹ thuật khác như kết nối tắt (skip connection) để kết nối các lớp xa nhau mà không mất thông tin. Phần trình bày phía dưới đây chúng em sẽ đi sâu hơn vào từng loại lớp để làm rõ cách thức hoạt động của chúng.

Lớp tích chập - convolution layer

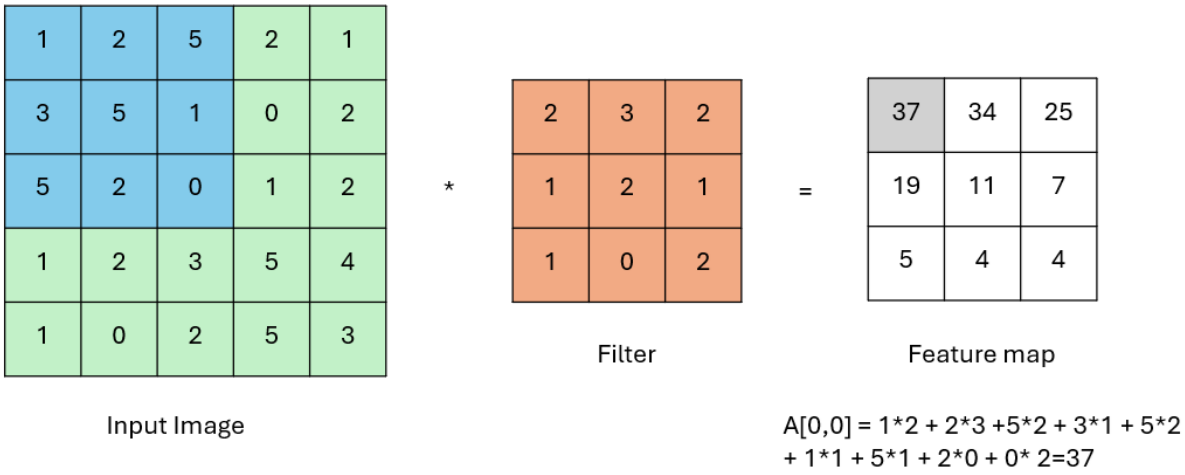
Lớp tích chập - convolution layer, cái tên của nó cũng nói lên tất cả sự quan trọng và không thể thay thế về vai trò của nó trong mạng CNN. Công dụng chính của lớp này là trích xuất các đặc trưng của dữ liệu.

Lớp tích chập bao gồm các thành phần là dữ liệu đầu vào, bộ lọc (filter, một số bài viết dùng kernel) và bản đồ đặc trưng (feature map). Bộ lọc là một ma trận nhỏ các trọng số, được sử dụng để trích xuất các đặc trưng từ dữ liệu đầu vào thông qua phép tích chập. Chúng thường có kích thước phổ biến là 3x3 hoặc 5x5. Các bộ lọc sẽ trượt theo chiều ngang và chiều dọc của dữ liệu đầu vào (ví dụ là hình ảnh), mỗi lần trượt bộ lọc sẽ nhảy sang vùng dữ liệu mới với bước nhảy (stride) s và thực hiện các phép tích chập. Kết quả thu được chính là feature map chứa các đặc trưng được trích xuất từ trên hình ảnh đầu vào. Ngoài ra, feature map cũng có thể được cộng thêm bias và đưa qua hàm kích hoạt để học thêm các mối quan hệ phức tạp hơn. Phép tích chập giữa ma trận đầu vào I và bộ lọc F với stride s có thể được biểu diễn bằng công thức sau (xem hình 3.2):

Trong đó:

- I là ma trận đầu vào kích thước $H \times W$,

- F là bộ lọc (hoặc kernel) kích thước $M \times N$,
- A là bản đồ đặc trưng kết quả sau khi áp dụng phép tích chập,
- i và j là chỉ số của phần tử trong ma trận feature map A,
- m và n là chỉ số của phần tử trong bộ lọc F,
- s là bước nhảy (stride) của bộ lọc mỗi lần di chuyển trên dữ liệu đầu vào.



Hình 3.2: Minh họa phép tích chập với ảnh kích thước 6x6, bộ lọc 3x3, bước nhảy 1.

$$A(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i \cdot s + m, j \cdot s + n) \cdot F(m,n)$$

Việc thực hiện phép tích chập như trên sẽ khiến kích thước của feature map khác với kích thước của dữ liệu đầu vào, để đảm bảo kích thước đầu ra như ý muốn người ta áp dụng kỹ thuật **zero padding**. Người ta sẽ thêm các số 0 xung quanh đường biên của ma trận trước khi áp dụng phép tích chập. Một số kiểu thêm đường viền phổ biến thường được sử dụng là valid padding, same padding và full padding.

Lớp gộp - pooling layer

Lớp gộp thường được sử dụng giữa các lớp tích chập để giảm kích thước không gian của feature map và giảm số lượng tham số trung bình, giúp mô hình trở nên đơn giản hơn. Bằng cách giữ lại các đặc trưng quan trọng và bỏ qua các chi tiết không quan trọng, pooling có thể cải thiện tính tổng quát hóa của mô hình.

Tương tự như lớp tích chập, lớp này cũng sẽ bộ lọc sẽ quét qua tất cả dữ liệu đầu vào. Tuy nhiên bộ lọc ở đây sẽ không có tham số nào mà sẽ sử dụng các hàm tổng hợp để tổng hợp lại thông tin của vùng dữ liệu tương ứng. Có hai loài hàm thường được sử dụng đại diện cho hai loại lớp gộp:

- **Max pooling**: khi bộ lọc di chuyển qua đầu vào, nó chọn giá trị lớn nhất từ vùng tương ứng để gửi đến mảng đầu ra. Phương pháp này thường được sử dụng nhiều hơn so với average pooling vì nó giữ lại các đặc trưng quan trọng nhất của vùng quét. Kết quả là mỗi vùng được giữ lại chỉ có một giá trị, đại diện cho giá trị lớn nhất trong vùng đó.

- **Average pooling**: khi bộ lọc di chuyển qua đầu vào, nó tính toán giá trị trung bình của các pixel trong vùng tương ứng và gửi giá trị trung bình này đến mảng đầu ra.

Mặc dù sử dụng lớp gộp có thể gây mất mất thông tin nhưng bằng cách giảm độ phức tạp của mô hình, nó sẽ giúp hạn chế việc overfitting hơn.

Lớp kết nối đầy đủ - fully-connected layer

Sau khi ảnh đã được xử lý qua nhiều lớp tích chập và gộp, mô hình đã học được các đặc trưng quan trọng của hình ảnh. Kết quả cuối cùng của lớp gộp thường là một tensor đa chiều, tensor này sẽ được làm phẳng thành một vector duy nhất, tức là tất cả các chiều sẽ được nối liền nhau. Lớp kết nối đầy đủ (FC) nhận vector này làm đầu vào và kết nối mỗi phần tử trong vector đó với tất cả các nơ-ron trong lớp này, giúp mô hình học các mối quan hệ toàn cục giữa các đặc trưng. Cuối cùng, đầu ra của lớp kết nối đầy đủ sẽ được đi qua một hàm kích hoạt mà thường là softmax

hoặc sigmoid để sản sinh ra xác suất dự đoán cuối cùng. Đây là cách mà mạng CNN hoàn thành quá trình phân loại ảnh dựa trên các đặc trưng mà nó đã học được từ ảnh đầu vào (xem hình 3.1).

3.1.3 Các kiến trúc mạng CNN tiêu biểu

Mạng CNN - Convolutional Neural Network luôn được các nhà nghiên cứu rất quan tâm và liên tục phát triển. Các kiến trúc mạng CNN tiêu biểu là:

- LeNet-5[9]
- AlexNet[7]
- VGGNet[12]
- ResNet[4]
- EfficientNet[13]

Trong đó, kiến trúc ResNet-32 được chúng em sử dụng để thực hiện thí nghiệm với CRUST.

3.2 Huấn luyện mô hình bằng phương pháp CRUST

3.2.1 Giới thiệu

Khó khăn của mạng nơ-ron khi huấn luyện với dữ liệu nhiễu

Mặc dù mạng nơ-ron, đặc biệt là mạng nơ-ron tích chập (CNN), thường đạt hiệu quả rất tốt trong các bài toán phân loại ảnh, nhưng chúng lại gặp phải nhiều khó khăn khi huấn luyện với dữ liệu có nhãn nhiễu. Một nghiên cứu về tổng quát hóa của học sâu đã chỉ ra rằng mạng nơ-ron dễ dàng học thuộc bất kỳ nhãn nào của dữ liệu, kể cả các nhãn ngẫu nhiên[15].

Một số thí nghiệm trong nghiên cứu này đã thay tất cả nhãn của dữ liệu bằng các nhãn ngẫu nhiên. Kết quả cho thấy rằng mô hình có thể đạt được lỗi huấn luyện bằng 0, nhưng độ chính xác trên tập kiểm thử không khác gì là chọn ngẫu nhiên, bởi không có sự tương quan giữa các nhãn

huấn luyện và nhãn thử nghiệm. Điều này chứng tỏ rằng mạng nơ-ron có khả năng ghi nhớ và học thuộc lòng các nhãn, dù chúng có thể không mang lại thông tin gì hữu ích cho việc tổng quát hóa.

Vấn đề này trở nên nghiêm trọng hơn khi dữ liệu nhiễu tăng lên. Dữ liệu nhiễu không chỉ làm giảm hiệu suất của mô hình trên tập kiểm thử mà còn gây ra hiện tượng overfitting dữ liệu huấn luyện. Mô hình học theo các nhãn sai lệch, dẫn đến dự đoán sai lầm và hiệu suất kém khi áp dụng vào thực tế.

Để huấn luyện mạng nơ-ron có độ chính xác cao, dữ liệu sạch là một yếu tố cực kỳ quan trọng. Tuy nhiên, việc thu thập một lượng dữ liệu lớn và sạch thì yêu cầu chi phí và thời gian rất cao. Các quy trình như thu thập, làm sạch và gán nhãn dữ liệu thủ công đòi hỏi sự can thiệp của con người và rất dễ xảy ra sai sót. Trong các hệ thống lớn, việc này không chỉ tốn kém mà còn khó duy trì chất lượng dữ liệu nhất quán.

Do đó, đã có nhiều phương pháp đề xuất để giải quyết vấn đề huấn luyện mô hình với dữ liệu nhiễu. Một trong những hướng tiếp cận nổi bật là sử dụng các kỹ thuật để tìm ra những mẫu dữ liệu chất lượng và loại bỏ các mẫu nhiễu. Trong khóa luận này, chúng em giới thiệu về phương pháp CRUST, một phương pháp lựa chọn ra tập con sạch và đạt hiệu suất rất tốt, lấy ý tưởng từ một số quan sát thú vị về sự phân bố của gradient. Phương pháp được xem như một giải pháp mới mẻ và hiệu quả để đối phó với dữ liệu nhiễu trong quá trình huấn luyện mạng nơ-ron.

Phương pháp CRUST

Phương pháp CRUST được đề xuất trong bài báo "Coresets for Robust Training of Neural Networks against Noisy Labels"[10]. Phương pháp này tập trung vào việc chọn ra các *coreset* từ tập dữ liệu gốc có nhãn nhiễu. Coreset là một tập con nhỏ của dữ liệu gốc, giữ lại những đặc điểm quan trọng nhất của toàn bộ tập dữ liệu. Trong ngữ cảnh của CRUST, mục tiêu là chọn ra các tập hợp con từ dữ liệu gốc có chứa nhãn nhiễu sao cho tập này không chỉ đại diện chính xác cho toàn bộ dữ liệu mà còn ít bị ảnh hưởng bởi nhiễu để huấn luyện mô hình, từ đó ngăn chặn mô hình học theo các nhãn sai lệch.

CRUST thực hiện lựa chọn coreset từ quan sát các điểm dữ liệu sạch sẽ gom cụm với nhau trên không gian gradient. Sau cùng, phương pháp CRUST không chỉ là một hướng đi mới mẻ trong lĩnh vực huấn luyện mô hình với nhãn nhiễu mà còn được chứng minh là hiệu quả thông qua nhiều thí nghiệm thực tế, cho thấy khả năng vượt trội trong việc giảm thiểu tác động của nhãn nhiễu so với các phương pháp trước đó.

3.2.2 Ý tưởng chính

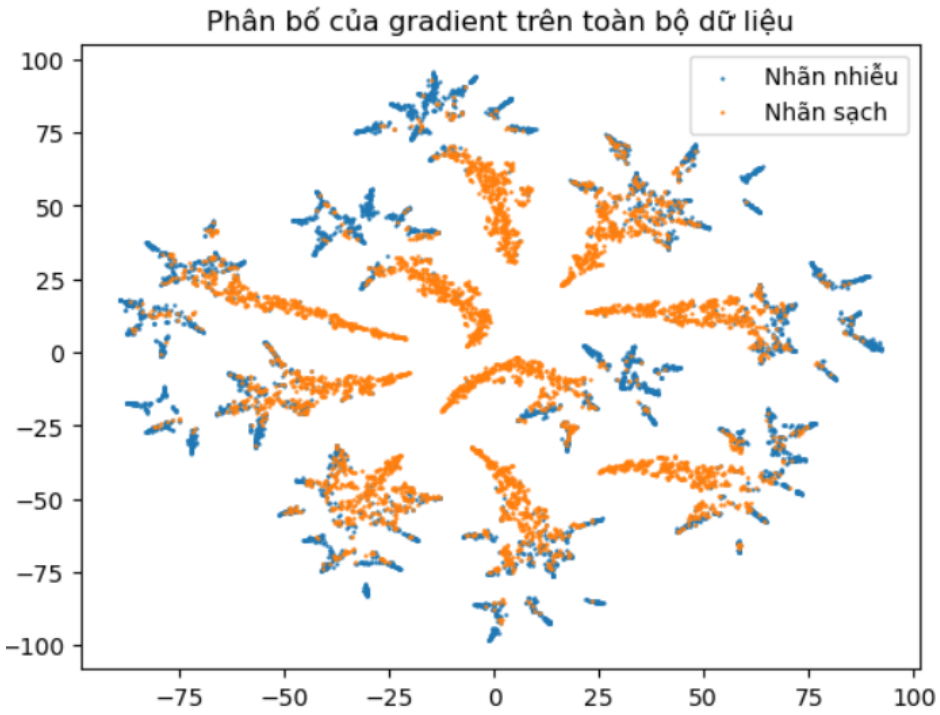
Gradient trong học máy hướng dẫn mô hình điều chỉnh các tham số của nó để giảm thiểu mất mát (sự khác biệt giữa dự đoán và giá trị thực tế). Ý tưởng về việc chọn coreset trong thuật toán CRUST bắt đầu từ những quan sát thú vị về nhãn sạch và nhãn nhiễu trên không gian gradient:

- Nhãn nhiễu có thể làm lạc lối quá trình tối ưu hóa. Khi huấn luyện với nhãn nhiễu, các gradient kéo mô hình ra xa hướng dự đoán thực sự. Trong đa số trường hợp, đặc biệt là đối với nhiễu đối xứng, mỗi nhãn nhiễu sẽ kéo mô hình ra xa một hướng khác nhau. Vì vậy, gradient của nhãn nhiễu sẽ phân bố rải rác trong không gian.

- Ngược lại, nhãn sạch cung cấp gradient nhất quán, chúng cùng thúc đẩy mô hình tập trung về hướng dự đoán chính xác. Vì vậy, gradient của dữ liệu sạch sẽ tập trung gần nhau trong không gian gradient.

Hình 3.3 thể hiện phân bố của gradient trên tập dữ liệu CIFAR với tỉ lệ nhiễu 50%. Mặc dù đã sử dụng t-SNE[14] để giảm chiều gradient về còn 2, sự phân bố có thể không hoàn toàn chính xác, nhưng nó cũng có thể phản ánh được một phần. Có thể dễ dàng nhìn thấy các điểm dữ liệu sạch phân bố gần nhau và các điểm dữ liệu nhiễu thường sẽ có sự phân bố tách biệt với các điểm dữ liệu sạch.

Trong bài toán huấn luyện mô hình trên dữ liệu nhiễu, để mô hình có thể dự đoán được trên dữ liệu sạch thì tỷ lệ nhãn sạch không được quá thấp (tỷ lệ chuyển đổi từ nhãn sạch sang nhãn khác do yếu tố nhiễu phải thấp hơn tỷ lệ nhãn sạch). Vì vậy, gradient của dữ liệu sạch sẽ là cụm lớn nhất. Phương pháp CRUST lần lượt xét các tập hợp dữ liệu có nhãn là c và xác định coreset dựa trên cụm gradient lớn nhất.



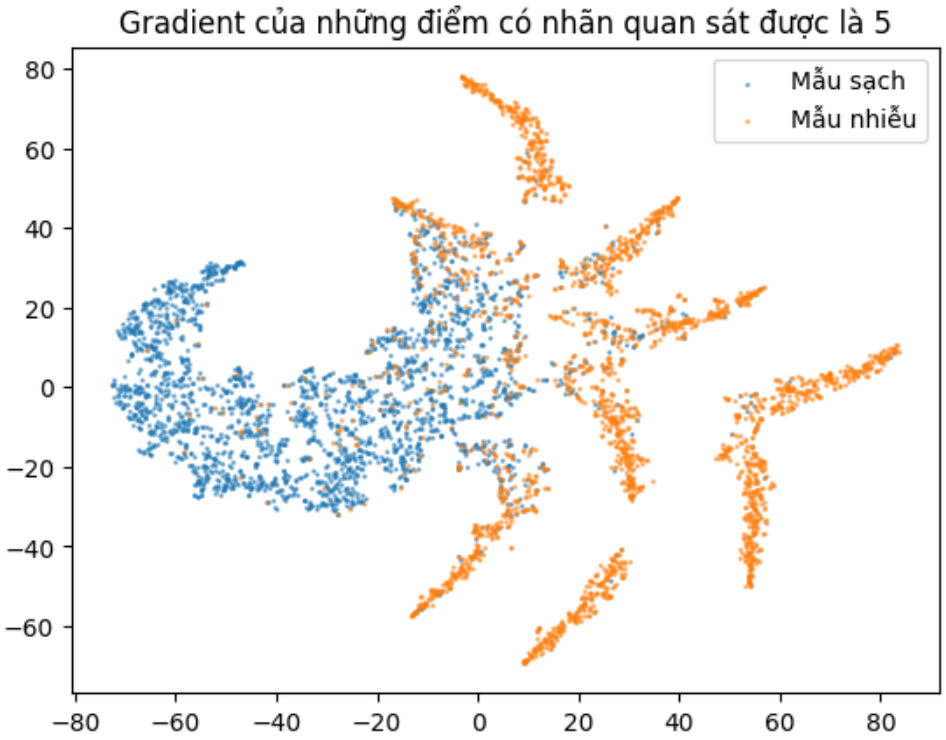
Hình 3.3: Phân bố của gradient sau khi sử dụng t-SNE để giảm chiều trên tập dữ liệu CIFAR-10 khi huấn luyện với mạng nơ-ron. Tập dữ liệu đã được cài đặt tỉ lệ nhiễu 50%, kiểu nhiễu đối xứng.

3.2.3 Các bước thực hiện

Phương pháp CRUST là một phương pháp lựa chọn dữ liệu nên sẽ có hai phần chính là lựa chọn dữ liệu sạch và huấn luyện trên dữ liệu đó. Như đã nói ở phần trước, CRUST xác định coreset bằng cách tìm cụm gradient lớn nhất. Trên thực tế, dữ liệu thường gom cụm theo lớp trước rồi mới gom cụm theo nhãn sạch và nhãn nhiễu. Vì vậy, để xác định coreset một cách chính xác hơn, CRUST chọn ra một tập hợp con các mẫu có nhãn là c và tìm coreset trên tập đấy. Hình 3.3 thể hiện phân bố gradient trên toàn bộ dữ liệu và hình 3.4 thể hiện phân bố gradient trên một lớp cụ thể.

Trong bài báo gốc, tác sử dụng hai cách là phân loại lớp theo nhãn quan sát được (nhãn nhiễu) và nhãn dự đoán để tìm coreset; tức là lọc ra các mẫu có nhãn nhiễu/nhãn dự đoán là c rồi mới tìm coreset. Tuy nhiên, trong phần thí nghiệm kết quả của nhãn nhiễu cao hơn nhiều so với nhãn dự đoán nên trong khóa luận này chúng em chỉ tập trung vào nhãn nhiễu (nhãn quan sát được).

Tổng kết, phương pháp CRUST sẽ chọn ra coreset từ dữ liệu sau đó



Hình 3.4: Phân bố gradient của nhiễu sym50, CIFAR 10 tại lớp 5.

huấn luyện trên coreset này. Thuật toán sẽ xét coreset của từng lớp trước sau đó mới gộp lại. Cụ thể, các bước của phương pháp được trình bày ở Algorithm 3 . Trong các phần trình bày tiếp theo chúng em sẽ giải thích rõ hơn về quá trình của mỗi bước làm.

3.2.4 Chọn coreset bằng thuật toán tìm cụm lớn nhất

Dữ liệu sạch thường gom thành một cụm riêng, dữ liệu nhiễu cũng gom thành các cụm nhỏ riêng, trong đó cụm của dữ liệu sạch thường là lớn nhất. Chúng ta có thể tìm cụm lớn nhất để xác định coreset. Để đạt được điều này, thuật toán dựa trên heuristic là: các điểm thuộc cụm có kích thước lớn thường sẽ có tổng khoảng cách đến các điểm dữ liệu khác là nhỏ, còn các điểm thuộc cụm có kích thước nhỏ thường sẽ có tổng khoảng cách đến các điểm dữ liệu khác là lớn. Khoảng cách này được tính bằng khoảng cách gradient giữa các điểm dữ liệu, cụ thể là:

$$d_{ij}(\mathbf{W}) = |\nabla\mathcal{L}(\mathbf{W}, x_i) - \nabla\mathcal{L}(\mathbf{W}, x_j)|_2, \tag{3.1}$$

trong đó $\nabla\mathcal{L}(\mathbf{W}, x_i)$ là gradient của hàm mất mát đối với điểm dữ liệu x_i.

Algorithm 3 Phương pháp CRUST

Đầu vào: Tập dữ liệu có chứa nhãn nhiễu $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ và số vòng lặp T.

Đầu ra: Tham số đầu ra của mô hình W^T.

for $\tau = 1, ...T$ **do**
  $S^\tau = \varnothing$.
  for $c \in \{1, ..., C\}$ **do**
    Chọn ra tập U_c^τ từ tập $\mathcal{D}$ là các phần tử có nhãn thuộc lớp c.
    Tính khoảng cách gradient d_{ij} giữa các điểm dữ liệu trong U_c^τ.
    Sử dụng thuật toán tìm cụm lớn nhất để tìm ra tập coreset S_c^τ theo lớp c.
  end for
  Gộp các tập coreset từng lớp thành tập coreset để huấn luyện S^τ.
  Huấn luyện mô hình trên coreset S^τ và cập nhật tham số W^T.
end for

Gradient của một điểm dữ liệu x_i với trọng số mạng $\mathbf{W}$ được biểu diễn là $\nabla\mathcal{L}(\mathbf{W}, x_i)$. Nó phản ánh độ nhạy của hàm mất mát đối với sự thay đổi của các trọng số mạng khi đưa điểm dữ liệu x_i vào.

Để tính toán khoảng cách gradient giữa các điểm dữ liệu, chúng ta cần thực hiện backpropagation trên toàn bộ tập dữ liệu. Trong các mạng nơ-ron, biến thiên của norm gradient chủ yếu được biểu diễn qua gradient của hàm mất mát đối với đầu vào của lớp cuối cùng của mạng nơ-ron[5]. Điều này có nghĩa là thay vì tính gradient cho toàn bộ các lớp của mạng nơ-ron, chúng ta có thể chỉ cần tính gradient đối với đầu vào của lớp cuối cùng.

Khi đó, khoảng cách gradient giữa hai điểm dữ liệu x_i và x_j được tính xấp xỉ như sau:

$$d_{ij} \approx \left\| \Sigma'_L(z_i^L)\nabla_i^L\mathcal{L} - \Sigma'_L(z_j^L)\nabla_j^L\mathcal{L} \right\|_2 \tag{3.2}$$

Trong đó $\Sigma'_L(z_i^L)\nabla_i^L\mathcal{L}$ là gradient của hàm mất mát $\mathcal{L}$ đối với đầu vào của lớp cuối cùng L cho điểm dữ liệu i.

Trong phần cài đặt, chúng em cũng sử dụng gradient đối với đầu vào của lớp cuối cùng của mạng nơ-ron để tính khoảng cách gradient của các điểm dữ liệu.

Thuật toán tìm cụm lớn nhất tương đối đơn giản. Đầu tiên, tính tổng

khoảng cách gradient từ một điểm đến tất cả các điểm còn lại để tạo ra ma trận khoảng cách gradient giữa các điểm. Vì mục tiêu là tìm các điểm thuộc cụm lớn, thế nên sẽ chọn ra các điểm có khoảng cách gradient đến các điểm dữ liệu khác là nhỏ nhất.

Thuật toán tìm cụm lớn nhất để chọn coreset từ tập dữ liệu gốc có thể được mô tả như sau:

- **Khởi tạo**:
  - Bắt đầu với tập rỗng $S = \varnothing$.
  - Xác định số điểm cần chọn k.
- **Lặp k lần**:
  - Chọn điểm i có tổng khoảng cách gradient đến các điểm còn lại là nhỏ nhất trong các điểm chưa chọn của tập đang xét.
  - Thêm điểm i vào tập S.
- **Kết quả**:
  - Trả về tập coreset S.

Sử dụng thuật toán tìm cụm lớn nhất để tìm coreset trên từng lớp, rồi gộp lại thành một coreset hoàn chỉnh. Sau đó có thể sử dụng coreset này để huấn luyện trên cho mô hình của mình.

3.2.5 Nhận xét và đánh giá

Phương pháp CRUST là một phương pháp hiệu quả trong lĩnh vực học sâu khi đối phó với vấn đề dữ liệu nhiễu. Ý tưởng chọn ra coreset gồm các điểm dữ liệu sạch qua các quan sát về sự phân bố của gradient trên tập dữ liệu có chứa nhiễu nhiễu là một cách tiếp cận sáng tạo, mới mẻ, vẫn còn tiềm năng lớn trong việc tiếp tục cải tiến và nâng cao hiệu suất.

Hiệu suất của CRUST đã được kiểm chứng thông qua nhiều thí nghiệm thực tế. Các thí nghiệm này cho thấy CRUST có khả năng vượt trội trong việc giảm thiểu tác động của nhãn nhiễu và cải thiện độ chính xác của mô

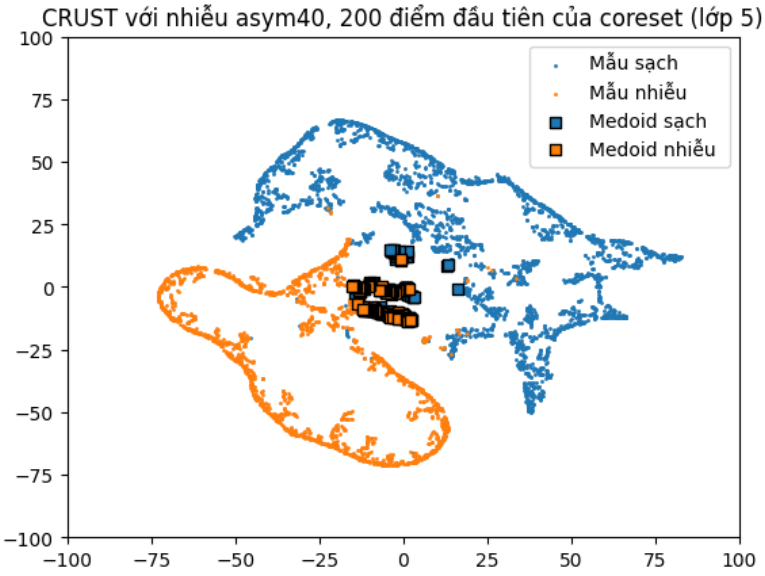
hình trên các tập kiểm thử. Phần này sẽ được trình bày chi tiết hơn ở chương 4.

Tuy nhiên, cách tiếp cận mới mẻ này vẫn còn rất nhiều điều khiến chúng ta nghi vấn về tính hiệu quả của phương pháp.

3.3 Cải tiến phương pháp CRUST

3.3.1 Vấn đề của CRUST

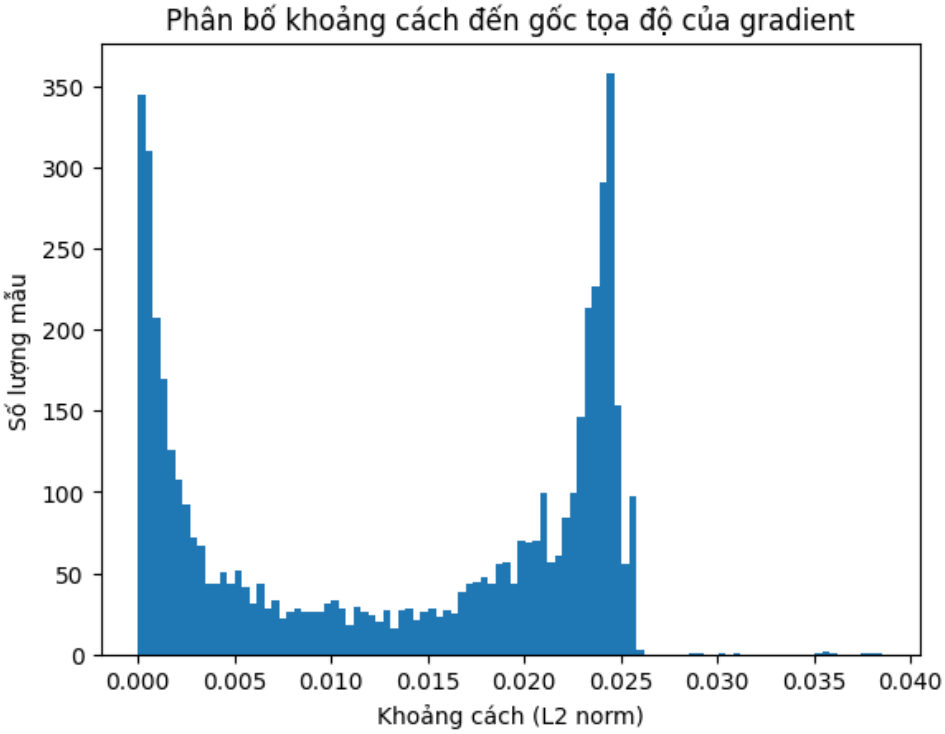
Hiện tại, phương pháp CRUST xác định cụm lớn nhất dựa trên heuristic tìm các điểm có tổng khoảng cách đến các điểm khác trong không gian gradient là nhỏ nhất. Cách làm này có một bất lợi là khi các cụm nhỏ hơn có kích thước khá lớn và nằm gần cụm lớn nhất, nó có thể chọn các điểm thuộc cụm nhỏ. Hình 3.5 minh họa cách chọn của thuật toán tìm cụm lớn nhất khi cụm nhiễu có kích thước lớn. Trong hình là gradient của nhiễu asym40 khi đưa về 2 chiều rồi mới bắt đầu chọn coreset. Lưu ý rằng đây chỉ là để minh họa để dễ hiểu hơn về cách hoạt động của thuật toán. Để chọn coreset đúng thì phải chọn ở không gian với số chiều gốc (trước khi giảm về 2D). Dù vậy, trên thực tế ở không gian với số chiều gốc thì vẫn xảy ra trường hợp cụm nhiễu có kích thước lớn và nằm gần cụm sạch tương tự như hình 3.5, điều này làm việc chọn các điểm dữ liệu sạch trở nên sai sót.



Hình 3.5: Thuật toán tìm cụm lớn nhất với gradient của nhiễu asym40 khi đã đưa về 2D.

3.3.2 Cải tiến CRUST bằng cách thay đổi thuật toán tìm cụm lớn nhất

Khi mô hình đã học được cách phân lớp cơ bản sau một số epoch thì độ lỗi của dữ liệu sạch thường sẽ nhỏ hơn độ lỗi của dữ liệu nhiễu. Điều này dẫn đến gradient của dữ liệu sạch thường sẽ gần gốc tọa độ hơn gradient của dữ liệu nhiễu (xem hình 3.6). Vì vậy, trong không gian gradient, ta có thể lấy ra các điểm dữ liệu thuộc cụm lớn (ứng với dữ liệu sạch trong không gian dữ liệu ban đầu) bằng cách ưu tiên các điểm dữ liệu gần gốc tọa độ.



Hình 3.6: Khoảng cách đến gốc tọa độ của gradient với dữ liệu nhiễu sym50, CIFAR 10.

Chúng em tiến hành cải tiến thuật toán CRUST bằng cách thay vì tính tổng khoảng cách từ một điểm đến tất cả các điểm còn lại thì sẽ tính khoảng cách từ điểm đó đến gốc tọa độ. Phương pháp này rất đơn giản nhưng lại mang đến hiệu quả rất cao. Thuật toán tìm cụm lớn nhất mới có thể được mô tả như sau:

- **Khởi tạo**:
  - Bắt đầu với tập rỗng $S = \varnothing$.

- Xác định số điểm cần chọn k.

- **Lặp k lần**:
  - Chọn điểm i có khoảng cách từ gradient đến gốc tọa độ là gần nhất trong các điểm chưa chọn của tập đang xét.
  - Thêm điểm i vào tập S.

- **Kết quả**:
  - Trả về tập coreset S.

Phương pháp CRUST sẽ lần lượt xét các tập hợp con có nhãn là c của dữ liệu, sau đó áp dụng thuật toán trên để tìm coreset rồi gộp lại, cuối cùng thực hiện huấn luyện mô hình trên coreset đã chọn.

Chương 4

Thí nghiệm

Chương này trình bày các thí nghiệm để đánh giá mô hình mạng nơ-ron với phương pháp lựa chọn dữ liệu CRUST ở chương 3. Đầu tiên, chúng em trình bày các thiết lập thí nghiệm, bao gồm: bộ dữ liệu, môi trường thí nghiệm, kiến trúc cụ thể của mạng nơ-ron, các siêu tham số. Tiếp theo, chúng em trình bày các thí nghiệm đã được thực hiện. Ở thí nghiệm 1, chúng em tiến hành so sánh kết quả cài đặt của khóa luận với bài báo gốc và thấy kết quả của chúng em tương tự với kết quả khi sử dụng mã nguồn công khai của bài báo gốc, nhưng nếu so với kết quả được công bố trong bài báo gốc thì vẫn có thấp hơn. Do đó, ở thí nghiệm 2, chúng em cố gắng cải thiện kết quả của phương pháp CRUST mà chúng em cài đặt bằng cách phân tích và tinh chỉnh các siêu tham số. Ở thí nghiệm 3, chúng em cố gắng cải thiện hơn nữa kết quả của phương pháp CRUST bằng đề xuất cải tiến đơn giản đã trình bày ở chương 3 đó là thay đổi thuật toán tìm cụm lớn nhất. Thí nghiệm 2 và thí nghiệm 3 là các thí nghiệm mở rộng ngoài bài báo gốc, nhằm cải thiện kết quả của phương pháp CRUST ở thí nghiệm 1.

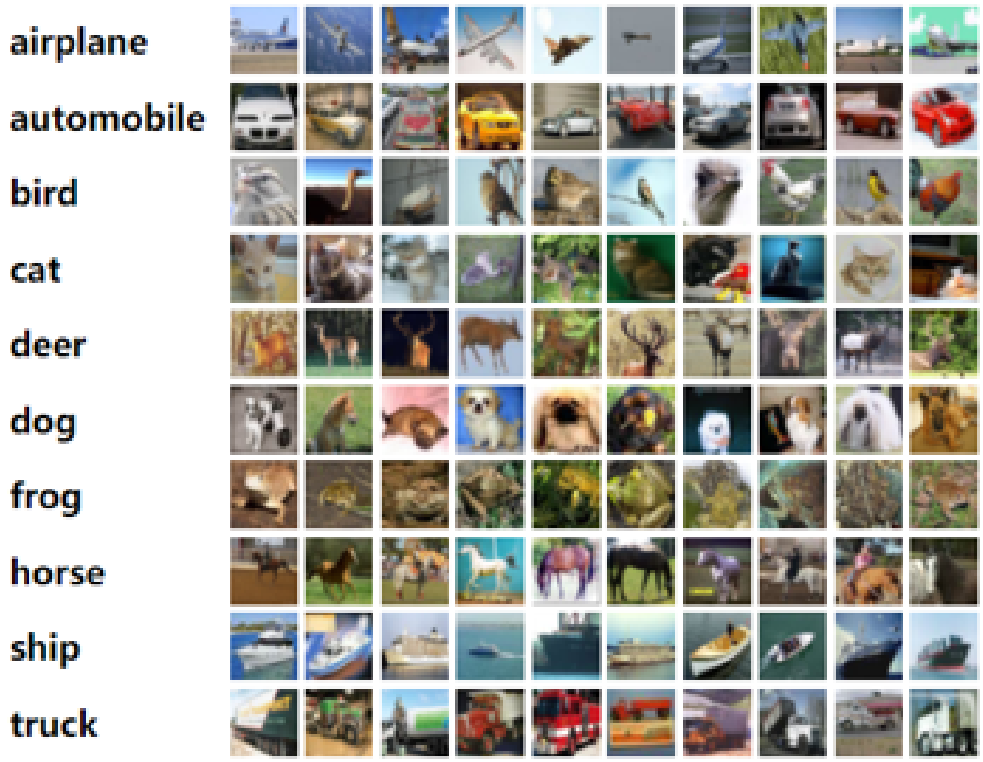
4.1 Thiết lập thí nghiệm

Bộ dữ liệu

Để có thể đánh giá hiệu suất của phương pháp CRUST khi huấn luyện mô hình với dữ liệu có nhãn nhiễu, chúng em đã sử dụng các bộ dữ liệu CIFAR-10, CIFAR-100. Các bộ dữ liệu được chọn nhằm mục đích kiểm tra khả năng của phương pháp trong việc xử lý dữ liệu chứa nhãn nhiễu trong các trường hợp khác nhau và so sánh với các phương pháp khác.

Đối với 2 bộ dữ liệu CIFAR-10 và CIFAR-100 thì: mỗi bộ dữ liệu đều có

60000 hình ảnh với kích thước 32x32 pixel. Trong đó bộ dữ liệu CIFAR-10 bao gồm 10 lớp khác nhau, mỗi lớp chứa 6000 hình ảnh. Còn bộ dữ liệu CIFAR-100 bao gồm 100 lớp khác nhau và mỗi lớp chứa 600 hình ảnh. Ở hình 4.1 là một số hình ảnh trên tập dữ liệu CIFAR-10. Chúng em chia 50000 ảnh cho quá trình huấn luyện và 10000 ảnh còn lại để cho quá trình validation. Cả 2 bộ dữ liệu này đều có nhãn sạch. Chính vì vậy, có thể dễ dàng điều chỉnh nhiễu để có một bộ dữ liệu với tỷ lệ nhiễu như mong muốn.



Hình 4.1: Một số hình ảnh trên tập dữ liệu CIFAR-10.

Cụ thể chúng em thực hiện tạo dữ liệu nhiễu[6] với hai kiểu nhiễu khác nhau: *nhiễu đối xứng (sym)* và *nhiễu bất đối xứng (asym).*

Nhiễu đối xứng xảy ra khi một nhãn đúng của dữ liệu bị thay thế ngẫu nhiên bằng một nhãn khác, với xác suất như nhau cho tất cả các nhãn. Ví dụ nếu nhãn gốc có các lớp A, B và C thì một nhãn A có thể bị đổi thành B và C với xác suất như nhau.

Nhiễu bất đối xứng xảy ra khi nhãn đúng của một dữ liệu luôn bị thay thế bởi một nhãn sai cụ thể. Ví dụ nếu nhãn gốc có các lớp A, B và C thì tất cả các nhãn nhiễu của A đều là nhãn B, hoặc tất cả nhãn nhiễu của A là nhãn C.

Từ bộ dữ liệu ban đầu, với tỷ lệ nhiễu và kiểu nhiễu cho trước, chúng em tiến hành tạo ra các bộ dữ liệu với các kiểu nhiễu và tỷ lệ nhiễu khác nhau, sau đó sử dụng các bộ dữ liệu này để đánh giá phương pháp trong từng trường hợp. Từ đó có thể đánh giá phương pháp một cách toàn diện hơn trong nhiều trường hợp khác nhau. Trong các thí nghiệm trên tập dữ liệu CIFAR, chúng em tiến hành thí nghiệm trên kiểu nhiễu đối xứng với các tỷ lệ nhiễu là 0.2, 0.5, 0.8. Đối với kiểu nhiễu bất đối xứng, chúng em tiến hành thí nghiệm với tỷ lệ nhiễu 0.4.

Các thiết lập khác

Để tiến hành thí nghiệm trên các bộ dữ liệu, chúng em huấn luyện mô hình trên T4 GPU của Kaggle. Chúng em cài đặt bằng ngôn ngữ Python và sử dụng mạng mô hình mạng nơ-ron để huấn luyện trên các bộ dữ liệu. Đối với các thí nghiệm trên các bộ dữ liệu CIFAR-10 và CIFAR-100 thì chúng em sử dụng mạng ResNet-32. Đây là mạng nơ-ron hiệu quả được sử dụng trong các bài toán phân loại ảnh.

Trước khi đưa dữ liệu vào quá trình huấn luyện, chúng em thực hiện một số bước xử lý ảnh trước đó[4]. Đầu tiên chúng em thực hiện thêm padding 4 pixel vào mỗi cạnh của ảnh gốc và cắt ngẫu nhiên một phần ảnh kích thước 32x32 từ ảnh gốc. Sau đó thực hiện lật ngẫu nhiên ảnh theo chiều ngang với xác suất 0.5. Việc áp dúng kỹ thuật tăng cường dữ liệu đơn giản này có thể giúp mô hình học tốt hơn.

Chúng em tiến hành các thí nghiệm trên tập dữ liệu CIFAR với mô hình ResNet-32[4] và các siêu tham số theo như tác giả của bài báo gốc đã tiến hành. Cụ thể, chúng em huấn luyện mô hình với 120 epoch, kích thước của mini-batch là 128. Chúng em sử dụng thuật toán để để cực tiểu hóa hàm chi phí là Stochastic Gradient Descent (SGD) với learning rate là 0.1, giảm tại epoch 80 và 100 với hệ số 10, momentum 0.9 và weight decay là 5×10^{-4}. Đối với CRUST, chúng em sử dụng coreset có kích thước là 50% so với tập dữ ban đầu.

Mô hình mạng nơ-ron ResNet-32

ResNet-32[4] là một phiên bản của mạng Residual Network (ResNet), được thiết kế để giải quyết vấn đề suy biến gradient (vanishing gradient) trong quá trình huấn luyện mạng nơ-ron sâu. ResNet đã chứng minh hiệu quả cao trong việc phân loại ảnh và đặc biệt giảm thiểu vấn đề suy biến gradient. Tập dữ liệu CIFAR có kích thước ảnh nhỏ (32x32), phù hợp với các mô hình có số lượng tham số vừa phải. ResNet-32 có kiến trúc phù hợp cho việc huấn luyện trên tập dữ liệu này, khiến nó trở thành baseline phổ biến trong các thí nghiệm liên quan đến CIFAR, cho phép dễ dàng so sánh kết quả giữa các phương pháp khác nhau. Mạng ResNet-32 thường được áp dụng trong các nghiên cứu về huấn luyện mạng nơ-ron với dữ liệu nhiễu, đặc biệt khi đánh giá phương pháp trên tập dữ liệu CIFAR.

4.2 Thí nghiệm 1: so sánh kết quả cài đặt của khóa luận với bài báo gốc

Chúng em thực hiện các thí nghiệm trên tập dữ liệu CIFAR để so sánh kết quả cài đặt của khóa luận với bài báo gốc. Mục đích chung của các thí nghiệm trên CIFAR là có thể hiểu rõ hiệu quả của phương pháp trên các dữ liệu có tỉ lệ nhiễu khác nhau và kiểu nhiễu khác nhau.

Trước hết, chúng em thực hiện thí nghiệm phương pháp CRUST trên tập dữ liệu CIFAR với chương trình chúng em tự cài đặt lại. Chúng em cài đặt lại dựa trên mã nguồn của bài báo gốc. Ngoài ra chúng em có thực hiện chạy lại phương pháp INCV. Qua các thí nghiệm này, giúp chúng em so sánh được hiệu quả giữa các phương pháp sử dụng, và kết quả chạy được thực tế so với kết quả đã đề cập trên bài báo gốc.

Sau khi thực hiện thí nghiệm, chúng em tiến hành so sánh kết quả của phương pháp mà chúng em tự cài đặt lại so với kết quả đã công bố ở bài báo của tác giả. Kết quả được thể hiện ở bảng 4.1.

Theo như kết quả có được ở bảng 4.1, có thể thấy rằng ý tưởng dựa trên sự phân bố của gradient để xác định các mẫu sạch, mẫu nhiễu là một điều hoàn toàn khả thi. Mặc dù có nhiều chỗ còn thấp hơn kết quả được công bố ở bài báo gốc, nhưng về tổng thể thì kết quả vẫn khá tốt. Về phần

	CIFAR-10				CIFAR-100		
	Sym		Asym		Sym		Asym
	20	50	80	40	20	50	40
INCV (Paper)	89.7	84.8	52.3	86.0	60.2	53.1	50.7
CRUST (Paper)	**91.1**	**86.3**	**58.3**	**88.8**	**65.2**	**56.4**	**53.0**
INCV	**89.9**	85.0	52.9	86.2	60.4	53.9	49.4
CRUST	85.4	**86.5**	36.5	78.3	63.9	54.5	**52.4**

Bảng 4.1: So sánh kết quả với bài báo gốc. Kết quả in đậm là kết quả tốt nhất hoặc xấp xỉ tốt nhất. Mỗi dòng của bảng đại diện cho một phương pháp; mỗi cột đại diện cho một bộ dữ liệu thuộc một loại nhiễu và có tỷ lệ nhiễu (%) tương ứng.

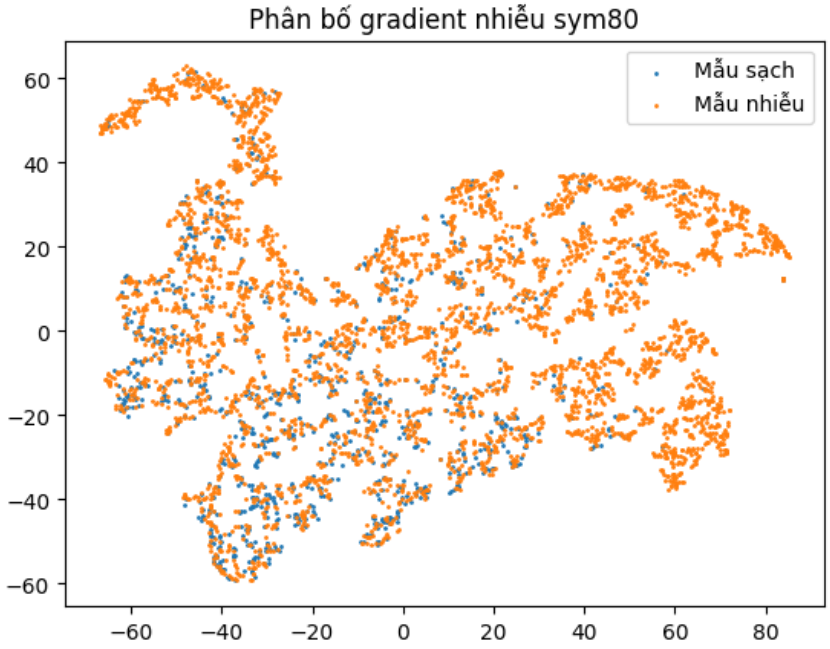
phương pháp INCV, đây vẫn là một phương pháp huấn luyện mạng nơ-ron với dữ liệu nhiễu rất tốt. Chúng em thực hiện chạy lại cài đặt theo phần mã nguồn công khai của phương pháp này thì cho vẫn cho kết quả cao và tương đương so với kết quả đã nêu trên bài báo của tác giả.

So sánh kết quả của thí nghiệm và kết quả được công bố ở bài báo gốc, chúng em rút ra các kết luận sau:

- **Kết quả sym 20 thấp hơn sym 50:** mặc dù cùng kiểu nhiễu, thậm chí tỷ lệ nhiễu còn thấp hơn nhưng kết quả ở sym 20 lại thấp hơn sym 50. Hình 4.3 cho thấy rằng tỷ lệ nhãn sạch trong coreset chọn được ở sym 20 tiệm cận 100%. Điều này chứng mình rằng việc sử dụng phương pháp CRUST trong việc chọn dữ liêu sạch là rất tốt. Tuy nhiên, với sym 20 thì đồng nghĩa có đến 80% dữ liệu sạch, trong khi đó coreset có kích thước mặc định là 50 (trong phần này khi nói kích thước coreset 50 thì có thể ngầm hiểu là 50% so với kích thước của dữ liệu), nên các điểm chọn có thể sẽ không bao gồm các điểm ở biên giữa "lãnh thổ" các lớp để thực hiện phân lớp ở cấp độ "mịn".

  Đối với nhiễu sym 50, sẽ có 50% dữ liệu sạch, khi đó coreset cũng có kích thước bằng 50 nên có thể bao phủ hết các điểm dữ liệu sạch này. Vì vậy nó sẽ có khả năng bao phủ được nhiều điểm gần biên giữa các lớp hơn (ít nhất là sẽ nhiều điểm gần biên hơn nhiễu sym 20), làm cho mô hình học cách phân lớp ở mức độ "mịn" tốt hơn dẫn đến hiệu suất cũng cao hơn.

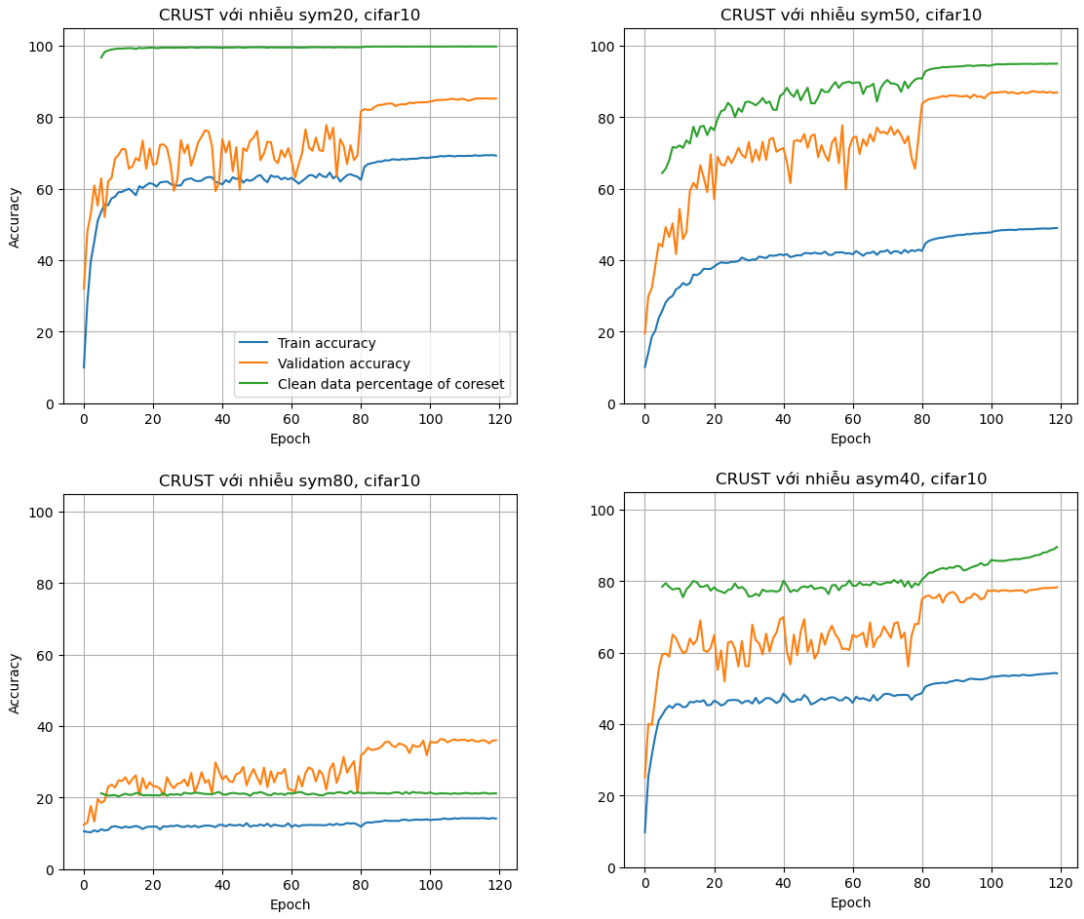
- **CRUST không hoạt động tốt trên sym 80:** thí nghiệm này thực hiện trên một tập dữ liệu có tỉ lệ nhiễu rất lớn, nên không thể tránh khỏi việc cho kết quả thấp. Nhưng kết quả mà chúng em đạt được khi thực hiện thấp hơn rất nhiều so với kết quả đạt được ở bài báo. Hình 4.3 cho thấy tỉ lệ nhãn chính xác trong coreset rất thấp, hầu như là chọn một cách ngẫu nhiên. Có thể thấy rằng phương pháp không hoạt động tốt và hầu như không thể tìm thấy dữ liệu sạch với tỉ lệ nhiễu này. Nguyên nhân dẫn đến việc này là do gradient của sym 80 cực kỳ hỗn loạn và không có dấu hiệu phân cụm dù mô hình đã học sau nhiều epoch. Hình 4.2 trực quan phân bố gradient của nhiễu sym 80 sau khi huấn luyện 120 epoch.



Hình 4.2: Phân bố gradient của nhiễu sym 80 của các mẫu có nhãn quan sát được là 5 sau khi huấn luyện 120 epoch.

- **Độ chính xác trên tập validation thấp hơn INCV ở một số thí nghiệm trên CIFAR-10 nhưng cao hơn trên CIFAR-100:** với CIFAR-10, CRUST có độ chính xác trên tập validation thấp hơn INCV ở sym 20 và asym 40, nhưng trên CIFAR-100 lại cao hơn. Với mức nhiễu sym 20 thì nguyên nhân dẫn đến điều này giống với vấn đề kết quả của sym 20 thấp hơn sym 50. Với mức nhiễu này, việc chọn

kích thước coreset bằng 50 là không đủ dữ liệu để mô hình học cách phân lớp ở mức độ "mịn". Tuy nhiên, đối với trường hợp CIFAR-100 thì có rất nhiều lớp, việc để cho mô hình học cách phân lớp ở mức độ thô đã là rất khó. Do đó, việc chỉ chọn các điểm phía trong lãnh thổ của lớp cũng sẽ giúp ích rất nhiều trong việc huấn luyện mô hình, vì vậy CRUST ở CIFAR-100 vẫn tốt hơn INCV. Với mức nhiễu là asym 40 của CIFAR-10, hình 4.3 cho thấy tỷ lệ dữ liệu sạch của coreset khá cao (gần 90%) nhưng độ chính xác trên tập validation khá thấp. Vì vậy, có thể có một nguyên nhân khác làm giảm độ chính xác của CRUST trên mức nhiễu này. Tuy nhiên với mức nhiễu này thì kích thước coreset mặc định là 50 cũng gần bằng với kích thước coreset tối ưu là 60 nên chưa thể xác định nguyên nhân giống như nhiễu sym 20 được. Trong thí nghiệm kế tiếp, chúng em sẽ thực hiện tinh chỉnh các siêu tham số mô hình và làm rõ hơn vấn đề này.



Hình 4.3: Biểu đồ độ chính xác của các mức nhiễu khác nhau trên CIFAR-10.

Tổng kết, kết quả cài đặt của chúng em nhìn chung có phần thấp hơn so với kết quả được công bố trong bài báo gốc. Tuy nhiên, kết quả cài đặt của chúng em tương tự với kết quả chạy lại mã nguồn được công bố của bài báo gốc (trong khóa luận chúng em cài đặt dựa trên mã nguồn của bài báo gốc).

4.3 Thí nghiệm 2: cải thiện kết quả của phương pháp CRUST bằng cách tinh chỉnh siêu tham số

Sự khác biệt về kết quả của thí nghiệm 1 có thể là do các siêu tham số mà chúng em sử dụng chưa hoàn toàn giống với bài báo gốc (chúng em đã sử dụng các siêu tham số theo như mô tả của bài báo gốc, nhưng có thể mô tả này chưa chính xác hoàn toàn). Do đó ở thí nghiệm này chúng em sẽ cố gắng cải thiện kết quả của thí nghiệm này bằng cách tinh chỉnh siêu tham số. Trong quá trình trực quan kết quả của CRUST với các mức độ nhiễu khác nhau (xem hình 4.3), chúng em nhận thấy rằng ở epoch 80, khi learning rate chuyển từ 0.1 sang 0.01 thì kết quả trên tập validation được tăng lên rất nhiều. Vì vậy, phương án đầu tiên chúng em đưa ra để cải thiện kết quả của CRUST là chọn learning rate ban đầu 0.01, mọi siêu tham số khác vẫn giữ nguyên; sau đó chọn kết quả tốt nhất giữa learning rate 0.1 và 0.01. Bên cạnh đó, việc phân tích kết quả của nhiễu sym 20 thấp hơn nhiễu sym 50 làm cho chúng em biết được tầm quan trọng của kích thước coreset. Do đó, ngoại trừ learning rate, chúng em cải thiện kết quả CRUST bằng cách kết hợp thêm với chọn kích thước coreset tốt nhất về mặt lý thuyết với loại nhiễu đang huấn luyện. Cụ thể, nếu đang huấn luyện với nhiễu sym 20 thì sẽ chọn kích thước coreset bằng 80, nếu là sym 80 sẽ chọn kích thước coreset là 20, còn nếu là asym 40 thì kích thước coreset là 60, riêng đối với sym 50 thì không cần thực hiện lại. Kết quả của thí nghiệm được trình bảy ở bảng 4.2

So sánh kết quả trước và sau khi tinh chỉnh siêu tham số của mô hình, chúng em rút ra các kết luận:

- Việc thay đổi learning rate ban đầu trong hầu hết trường hợp đều không giúp tăng hiệu suất của mô hình, ngoại trừ đối với sym 80.

	CIFAR-10				CIFAR-100		
	Sym		Asym		Sym		Asym
	20	50	80	40	20	50	40
INCV (Paper)	89.7	84.8	52.3	86.0	60.2	53.1	50.7
CRUST (Paper)	**91.1**	**86.3**	**58.3**	**88.8**	**65.2**	**56.4**	**53.0**
INCV	**89.9**	85.0	52.9	86.2	60.4	53.9	49.4
CRUST coreset size 50, learning rate 0.1	85.4	**86.5**	36.5	78.3	63.9	54.5	**52.4**
CRUST coreset size 50, learning rate tuned	85.4	**86.5**	40.2	78.3	63.9	54.5	**52.4**
CRUST coreset size tuned, learning rate tuned	**90.4**	**86.5**	40.2	78.3	**64.4**	54.5	**52.4**

Bảng 4.2: So sánh kết quả thí nghiệm của các phương pháp cũ và phương pháp CRUST sau khi điều chỉnh siêu tham số. Kết quả in đậm là kết quả tốt nhất hoặc xấp xỉ tốt nhất. Mỗi dòng của bảng đại diện cho một phương pháp; mỗi cột đại diện cho một bộ dữ liệu thuộc một loại nhiễu và có tỷ lệ nhiễu (%) tương ứng.

Mặc dù hiệu suất ở mức nhiễu này có tăng, nhưng nếu so với kết quả bài báo thì vẫn còn thua rất nhiều.

- Việc thay đổi kích thước coreset mang lại hiệu quả, nhưng chỉ tốt khi tỷ lệ nhiễu thấp. Cụ thể, ở mức nhiễu sym 20 của cả CIFAR-10 và CIFAR-100 đều có kết quả tăng. Điều này chứng minh rằng nhận định: kích thước coreset 50 không đủ để mô hình học cách phân lớp ở mức độ "mịn" với nhiễu sym 20 là đúng. Bên cạnh đó, khi xét với mức nhiễu là asym 40, độ chính xác của mô hình không tăng thêm. Điều này cũng dễ hiểu khi kích thước coreset mặc định là 50 cũng không khác với kích thước coreset tối ưu là 60 quá nhiều. Như vậy, nguyên nhân kết quả của mức nhiễu asym 40 trên CIFAR-10 bị thấp trong thí nghiệm trước vẫn chưa thể giải thích được, nhưng không phải do kích thước coreset, cũng không phải do learning rate. Cuối cùng, với mức nhiễu nặng như sym 80, việc giảm coreset không giúp CRUST chọn nhiều nhãn sạch hơn mà còn làm giảm dữ liệu huấn luyện nên cũng không tăng hiệu suất mô hình.

4.4 Thí nghiệm 3: cải thiện kết quả của phương pháp CRUST bằng cách thay đổi thuật toán tìm cụm lớn nhất

Chúng em đã trình bày ý tưởng và các bước thực hiện để cải tiến phương pháp CRUST trong mục 3.3. Trong phần dưới đây chúng em sẽ trình bày kết quả thí nghiệm của phương pháp cải tiến này. Trước hết, chúng em thực hiện thí nghiệm CRUST cải tiến với kích thước coreset là 50, giữ nguyên các siêu tham số khác so với thí nghiệm gốc của bài báo (ngoại trừ learning rate) rồi chọn kết quả tốt nhất giữa learning rate 0.1 và 0.01. Tiếp theo, chúng em kết hợp thêm với việc chọn ra coreset có kích thước phù hợp nhất về mặt lý thuyết tương tự như ở thí nghiệm 2. Kết quả thí nghiệm được trình bày trong bảng 4.3

So sánh kết quả trước và sau khi CRUST sử dụng thuật toán tìm cụm lớn nhất mới, chúng em có các kết luận sau:

- Với mức kích thước coreset 50, CRUST cải tiến giúp tăng mạnh hiệu suất mô hình (so với các thí nghiệm chúng em cài đặt) ở các mức nhiễu sym 80 - CIFAR-10 (6.8%), asym 40 - CIFAR-10 (8%), sym 50 - CIFAR-100 (7.2%). Thậm chí với mức nhiễu sym 50 - CIFAR-100 thì kết quả vượt xa so với kết quả công bố ở bài báo gốc. Các mức nhiễu khác thì thấp hơn bản trước cải tiến, nhưng không chênh lệch nhiều (so với coreset 50).

- Với kích thước coreset tối ưu nhất, CRUST cải tiến đạt hiệu quả tốt nhất, đa số các mức nhiễu đều có kết quả tăng khi chọn coreset phù hợp hơn, ngoại trừ mức nhiễu asym 40 nhưng với mức nhiễu này thì coreset kích thước mặc định là 50 và kích thước tốt nhất là 60 cũng không khác gì nhiều. Kết quả của CRUST cải tiến trong điều kiện tối ưu nhất rất tốt, gần như tất cả các trường hợp đều cho kết quả tốt hơn phương pháp CRUST bản cũ do chúng em cài đặt; ngoại trừ mức nhiễu sym 20 - CIFAR-10 và asym 40 - CIFAR-100 thấp hơn nhưng cũng không thấp hơn quá nhiều. Nếu so sánh kết quả với bài báo gốc thì cũng có một số mức nhiễu đạt kết quả tốt hơn như sym

	CIFAR-10				CIFAR-100		
	Sym		Asym		Sym		Asym
	20	50	80	40	20	50	40
INCV (Paper)	89.7	84.8	52.3	86.0	60.2	53.1	50.7
CRUST (Paper)	**91.1**	**86.3**	**58.3**	**88.8**	65.2	56.4	**53.0**
INCV	89.9	85.0	52.9	86.2	60.4	53.9	49.4
CRUST coreset size 50, learning rate 0.1	85.4	**86.5**	36.5	78.3	63.9	54.5	**52.4**
CRUST coreset size 50, learning rate tuned	85.4	**86.5**	40.2	78.3	63.9	54.5	**52.4**
CRUST coreset size tuned, learning rate tuned	**90.4**	**86.5**	40.2	78.3	64.4	54.5	**52.4**
CRUST improved coreset size 50, learning rate tuned	84.5	**86.9**	47.0	86.3	62.6	**61.7**	51.2
CRUST improved coreset size tuned, learning rate tuned	**90.3**	**86.9**	48.4	86.3	**67.4**	**61.7**	51.2

Bảng 4.3: So sánh kết quả thí nghiệm của các phương pháp cũ và phương pháp CRUST sau khi sử dụng thuật toán mới để chọn coreset. Kết quả in đậm là kết quả tốt nhất hoặc xấp xỉ tốt nhất. Mỗi dòng của bảng đại diện cho một phương pháp; mỗi cột đại diện cho một bộ dữ liệu thuộc một loại nhiễu và có tỷ lệ nhiễu (%) tương ứng.

50 - CIFAR-10 (0.6%), sym 20 - CIFAR-100 (2.2%), thậm chí là vượt xa như sym 50 - CIFAR-100 (5.3%).

Tổng kết, sau khi cải tiến bằng cách thay đổi thuật toán tìm coreset, CRUST chọn được nhiều dữ liệu sạch hơn, giúp hiệu suất của mô hình khi sử dụng phương pháp này đạt kết quả cao hơn với dữ liệu nhiễu.

Chương 5

Kết luận và hướng phát triển

5.1 Kết luận

Dữ liệu nhiễu xuất hiện rất nhiều trong các bộ dữ liệu thực tế. Khóa luận đã thực hiện tìm hiểu phương pháp giải quyết bài toán học có giám sát với dữ liệu nhiễu. Đây là một vấn đề quan trọng, đáng quan tâm và mang lại nhiều ý nghĩa quan trọng nếu giải quyết được. Phạm vi của khóa luận tập trung vào huấn luyện mô hình mạng nơ-ron. Chúng em đã trình bày chi tiết các nội dung liên quan đến mạng nơ-ron, nền tảng của nhiều phương pháp học máy hiện đại. Để giải quyết bài toán học có giám sát với dữ liệu nhiễu, chúng em đã giải quyết theo hướng sử dụng các phương pháp lựa chọn dữ liệu. Kết quả của các thí nghiệm cho thấy các phương pháp đều hoạt động tốt khi huấn luyện với dữ liệu nhiễu.

Phương pháp chính mà chúng em tập trung nghiên cứu và tìm hiểu sâu là phương pháp CRUST. Đây là một phương pháp với một ý tưởng mới mẻ, dựa vào các quan sát về sự phân bố của gradient để xác định dữ liệu có nhãn sạch và dữ liệu có nhãn nhiễu. Các điểm dữ liệu sạch gom lại thành 1 cụm với nhau, các điểm dữ liệu nhiễu cũng gom lại thành các cụm với nhau trong không gian gradient. Do lượng dữ liệu sạch lớn nhất nên sẽ gom thành cụm lớn nhất trong không gian gradient. Kết quả của cách làm dựa theo quan sát này cho một hiệu suất rất tốt.

Tuy nhiên trong quá trình cài đặt lại thuật toán và thực hiện các thí nghiệm, có một số kết quả thí nghiệm không đạt được kết quả như trong bài báo gốc mà tác giả đã thực hiện. Nhưng cũng nhờ điều đó, chúng em tiếp tục thực hiện thêm một số thí nghiệm mở rộng để tìm hiểu nguyên nhân và nhận ra một số vấn đề của cách làm này. Sau đó, chúng em cải tiến phương pháp bằng cách thay đổi một thuật toán khác để tìm coreset và thu được một kết quả tốt hơn.

5.2 Hướng phát triển

Phương pháp CRUST có cách tiếp cận mới mẻ và đơn giản, nên vẫn còn rất nhiều tiềm năng để tiếp tục cải tiến nâng cao hiệu suất. Về cách chọn coreset, hiện tại phương pháp được cài đặt bằng một thuật toán rất đơn giản để tìm cụm lớn nhất. Thuật toán này giúp tìm coreset dễ dàng, nhưng đây vẫn chưa là cách làm tối ưu, chúng ta vẫn có cách khác để hiệu quả hơn.

CRUST là một phương pháp lựa chọn dữ liệu, kết quả cuối cùng là chọn ra tập con có dữ liệu sạch. Bước kế tiếp là đưa vào một mô hình mạng nơ-ron và huấn luyện bình thường mà không bị ảnh hưởng, hay cần thêm sự can thiệp và điều chỉnh nào khác. Chính vì sự đơn giản này, chúng ta có thể dễ dàng kết hợp phương pháp CRUST với bất kỳ phương pháp, mô hình huấn luyện khác nhau để tăng sự hiệu quả.

Cách mà bài toán thực hiện là chọn ra một coreset gồm các điểm dữ liệu sạch và đại diện cho dữ liệu. Với ý tưởng này, chúng ta có thể hướng tới giải quyết một bài toán khác là chọn một tập con nhỏ hơn từ một tập dữ liệu rất lớn, sao cho hiệu quả khi huấn luyện trên tập con vẫn được đảm bảo. Giải quyết bài toán này có thể giúp chúng ta tiết kiệm thời gian khi huấn luyện trên tập dữ liệu rất lớn nhưng trong dữ liệu có nhiều điểm tương đồng với nhau.

Ngoài ra, cách tiếp cận bài toán dựa trên sự phân bố gradient của dữ liệu là một cách tiếp cận rất hay. Nên có nhiều nghiên cứu hơn về cách tiếp cận này trong việc giải quyết các bài toán liên quan đến vấn đề dữ liệu. Cũng có thể cách tiếp cận này không chỉ giải quyết tốt bài toán học có giám sát với dữ liệu nhiễu mà nó cũng có thể mang lại giá trị nhất định nào đó trong các bài toán khác.

Tài liệu tham khảo

[1] Avrim Blum and Tom Mitchell. "Combining labeled and unlabeled data with co-training". In: *Proceedings of the eleventh annual conference on Computational learning theory*. 1998, pp. 92–100.

[2] Pengfei Chen et al. "Understanding and Utilizing Deep Neural Networks Trained with Noisy Labels". In: *International Conference on Machine Learning*. 2019, pp. 1062–1070.

[3] Bo Han et al. "Co-teaching: Robust training of deep neural networks with extremely noisy labels". In: *NeurIPS*. 2018, pp. 8535–8545.

[4] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[5] Angelos Katharopoulos and François Fleuret. "Not all samples are created equal: Deep learning with importance sampling". In: *International conference on machine learning*. PMLR. 2018, pp. 2525–2534.

[6] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[8] Anders Krogh and John A Hertz. "A simple weight decay can improve generalization". In: *Proc, NeurIPS*. 1992, pp. 950–957.

[9] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[10] Baharan Mirzasoleiman, Kaidi Cao, and Jure Leskovec. "Coresets for Robust Training of Neural Networks against Noisy Labels". In: *Advances in Neural Information Processing Systems* 33 (2020).

[11] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. "Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks". In: *Proceedings of the 35th Conference on Neural Information Processing Systems Track on Datasets and Benchmarks*. Dec. 2021.

[12] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014).

[13] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[14] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[15] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *Communications of the ACM* 64 (2016). DOI: 10.1145/3446776.