

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
NHIỀU BẰNG PHƯƠNG PHÁP LỰA
CHỌN DỮ LIỆU

KHÓA LUẬN TỐT NGHIỆP CỦ 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
NHIỀU BẰNG PHƯƠNG PHÁP LỰA
CHỌN DỮ LIỆU**

KHÓA LUẬN TỐT NGHIỆP CỦ 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 đợt 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

Lời cảm ơn	i
Danh sách hình	iv
Danh sách bảng	vi
1 Giới thiệu	1
1.1 Đặt vấn đề	1
1.2 Bài toán học có giám sát với dữ liệu nhiễu	3
1.3 Các phương pháp giải quyết bài toán	4
2 Kiến thức nền tảng	6
2.1 Mô hình mạng nơ-ron đơn giản	6
2.1.1 Dạng mô hình	6
2.1.2 Huấn luyện mô hình bằng Gradient Descent	11
2.2 Phương pháp chống quá khớp Weight Decay và Validation	16
2.3 Phương pháp lựa chọn dữ liệu INCV	17
2.3.1 Giới thiệu	17
2.3.2 Nguyên lý hoạt động INCV	18
2.3.3 Chi tiết thuật toán	19
3 Mô hình mạng nơ-ron với phương pháp lựa chọn dữ liệu CRUST	22
3.1 Dạng mô hình	22
3.1.1 Giới thiệu	23
3.1.2 Các thành phần cơ bản của mạng tích chập	23
3.1.3 Các kiến trúc mạng CNN tiêu biểu	27
3.2 Huấn luyện mô hình bằng phương pháp CRUST	28
3.2.1 Giới thiệu	28
3.2.2 Ý tưởng chính	29

3.2.3	Các bước thực hiện	30
3.2.4	Chọn tập con bằng thuật toán k-medoids	32
3.2.5	Nhận xét và đánh giá	36
4	Thí nghiệm	40
4.1	Thiết lập thí nghiệm	40
4.2	Thí nghiệm 1: so sánh kết quả cài đặt của khóa luận với bài báo gốc	42
4.3	Thí nghiệm 2: phân tích và cải thiện kết quả của phương pháp CRUST ở thí nghiệm 1	46
5	Kết luận và hướng phát triển	55
5.1	Kết luận	55
5.2	Hướng phát triển	56
Tài liệu tham khảo		58

Danh sách hình

1.1	Một số mẫu dữ liệu bị gán nhãn sai trên tập dữ liệu ImageNet	2
2.1	Fully-Connected FeedForward Neural Network với 2 lớp ẩn cho bài toán phân lớp.	7
2.2	Hình minh họa cho một nơ-ron trong học máy.	8
2.3	Hình minh họa hàm lỗi huấn luyện J có dạng parabol	13
2.4	Learning rate thấp và cao	15
2.5	K-Fold Cross-Validation	19
3.1	Mạng CNN đơn giản cho bài toán nhận dạng chữ số viết tay trên tập dữ liệu MNIST	24
3.2	Minh họa phép tích chập với ảnh kích thước 6x6, bộ lọc 3x3, bước nhảy 1	26
3.3	Phân bố của gradient sau khi sử dụng t-SNE để giảm chiều trên tập dữ liệu CIFAR-10	31
3.4	Phân bố của gradient của các mẫu có nhãn nhiễu là 5 sau khi sử dụng t-SNE để giảm chiều trên tập dữ liệu CIFAR-10	37
3.5	Phân bố của gradient của các mẫu có nhãn nhiễu là 5 và nhãn dự đoán là 5 sau khi sử dụng t-SNE để giảm chiều .	38
4.1	Một số hình ảnh trên tập dữ liệu CIFAR-10	41
4.2	Biểu đồ độ chính xác của mô hình khi huấn luyện trên tập CIFAR với sym 80	45
4.3	Phân bố gradient của nhiễu đối xứng 50%	47
4.4	So sánh kết quả huấn luyện nhiễu đối xứng 20% và 50% trên CIFAR-10	49
4.5	So sánh kết quả huấn luyện nhiễu đối xứng 20% theo phương pháp ban đầu và sau khi cải tiến CIFAR-10	50
4.6	So sánh sự không đồng đều giữa số lượng mẫu của lớp trong coreset ở mức nhiễu đối xứng 50%	52

4.7	Phân bố gradient của các loại nhiễu khác nhau trên CIFAR-10 tại epoch 119	53
4.8	Label Accuracy của các nhãn trong coreset với mức nhiễu asym 40 trên tập CIFAR-10 và CIFAR-100	54

Danh sách bảng

4.1 So sánh kết quả với bài báo gốc	44
---	----

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.

Tuy nhiên, chất lượng dữ liệu vẫn 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)[17] chỉ ra rằng tỷ lệ nhãn nhiễu trong tập dữ liệu ImageNet là khoảng 6%. Nhưng 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,

			
ImageNet given label: <u>tick</u> Cleanlab guessed: yellow garden spider MTurk consensus: yellow garden spider	ImageNet given label: <u>coyote</u> Cleanlab guessed: dingo MTurk consensus: dingo ID: 00012230	ImageNet given label: <u>weasel</u> Cleanlab guessed: badger MTurk consensus: badger ID: 00031073	ImageNet given label: <u>sloth bear</u> Cleanlab guessed: brown bear MTurk consensus: brown bear ID: 00047458

Hình 1.1: Một số mẫu dữ liệu bị gán nhãn sai trên tập dữ liệu ImageNet (Nguồn: labelerrors.com)

giảm độ chính xác dự đoán, tăng nguy cơ sai sót và ảnh hưởng đến khả năng khai 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 nhiễu 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.

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 $\{\tilde{y}_i\}_{i=1}^n$.
- Yêu cầu: tìm hàm $f : \mathbb{R}^d \rightarrow \mathbb{R}$ có khả năng dự đoán nhãn thật $\tilde{y} \in \mathbb{R}$ 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 sai các đặc trưng và giảm độ chính xác dự đoán. Thứ hai, mạng nơ-ron có khả năng ghi nhớ bất kỳ nhãn nào (thậm chí ngẫu nhiên) của dữ liệu[23]. Do đó, nếu không có các biện pháp xử lý phù hợp, mô hình có thể học sai các đặc trưng từ dữ liệu nhiễu và dẫn đến hiện tượng overfitting, giảm khả năng tổng quát hóa trên dữ liệu mới. Thứ ba, dữ liệu có nhãn nhiễu làm giảm khả năng của mô hình trong việc học các mẫu chính xác từ dữ liệu. Mô hình có thể học các đặc trưng không chính xác từ dữ liệu có nhãn nhiễu, dẫn đến việc dự đoán sai khi áp dụng trên dữ liệu mới.

Ý nghĩa của bài toán

Tuy nhiên, 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 nhiều 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 nhiều 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 nhiều là một vấn đề lớn.

Như vậy, giải quyết bài toán học có giám sát với dữ liệu nhiều 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 nhiều 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[12]. Đây là một kỹ thuật regularization giúp ngăn chặn mô hình học theo các nhãn nhiều 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 đặc trưng không chính xác 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)[4]. 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 mờ 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 các 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 cực kỳ tốt đó là phương pháp CRUST (Mirzasoleiman et al., 2020)[16]. 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. Đây là một cách tiếp cận khá mới mẻ nhưng cũng mang lại hiệu quả rất tốt. Đây cũng là phương pháp mà chúng em chọn để nghiên cứu và tìm hiểu sâu.

Trong khóa luận này, chúng em sẽ tập trung tìm hiểu và cài đặt phương pháp CRUST. Việc tìm hiểu và cài đặt phương pháp này không chỉ giúp chúng em hiểu rõ hơn về phương pháp lựa chọn dữ liệu mà còn hiểu các cơ sở lý thuyết để một mô hình hoạt động tốt với dữ liệu nhiễu.

Chương 2

Kiến thức nền tảng

Trong chương này, chúng em sẽ trình bày các kiến thức nền tảng quan trọng phục vụ cho việc nghiên cứu về học có giám sát bằng phương pháp lựa chọn dữ liệu. Đầu tiên, chúng em sẽ giới thiệu về mô hình mạng nơ-ron đơn giản, là nền tảng của nhiều phương pháp học máy hiện đại, cũng là nền tảng để trình bày về dạng mô hình/kiến trúc sử dụng trong phương pháp CRUST. Sau đó, chúng em sẽ trình bày về phương pháp Weight Decay và Validation, hai kỹ thuật cơ bản nhưng không kém quan trọng để cải thiện hiệu suất của mô hình khi huấn luyện, đặc biệt với dữ liệu nhiễu. Cuối cùng, chúng em sẽ giới thiệu về phương pháp lựa chọn dữ liệu INCV, một phương pháp hiệu quả để tối ưu hóa mô hình học có giám sát. Chương này cung cấp những kiến thức cần thiết để hiểu rõ về phương pháp CRUST được trình bày trong chương tiếp theo.

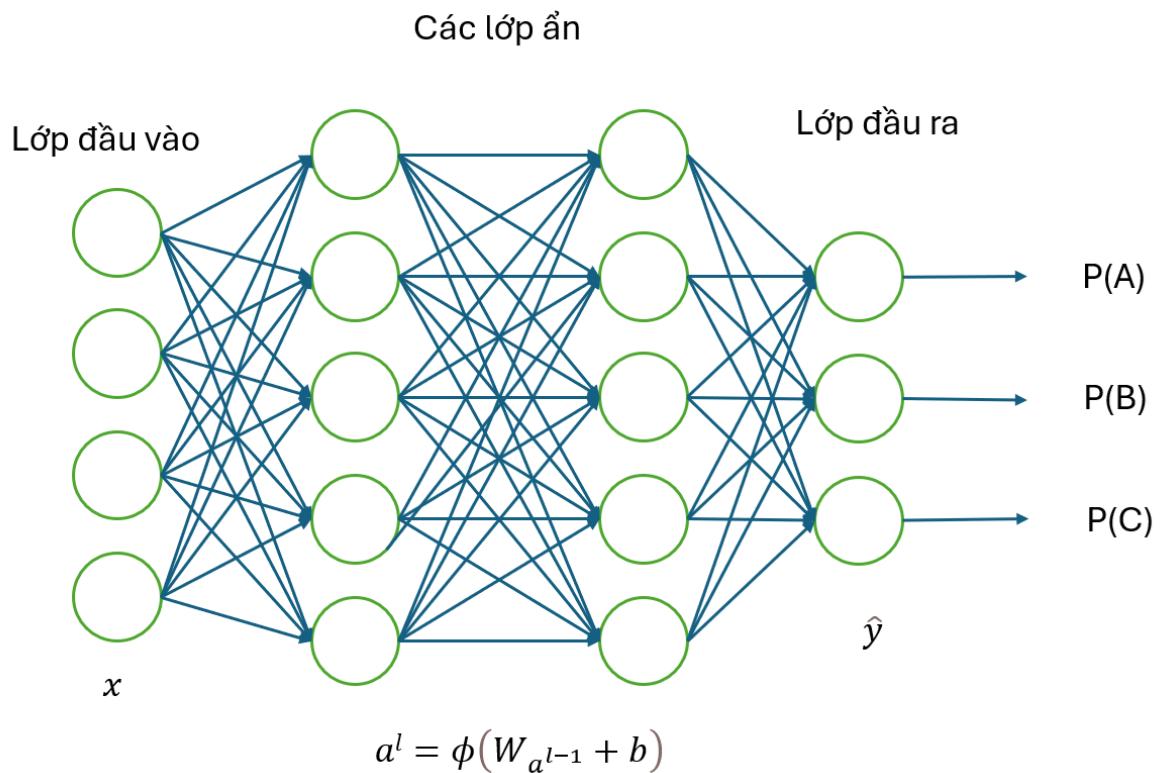
2.1 Mô hình mạng nơ-ron đơn giản

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/khiến trúc đơn giản nhất là mạng FCFFNN.

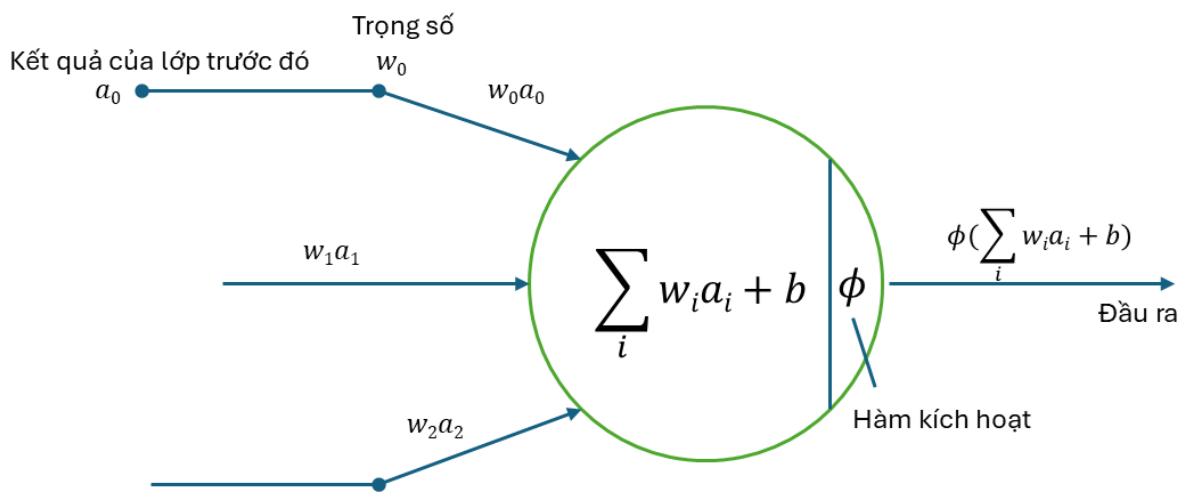
Fully-Connected FeedForward Neural Network (FCFFNN) là một loại mạng nơ-ron nhân tạo (Artificial Neural Network - ANN) trong đó các nơ-ron ở mỗi lớp (layer) đều được kết nối với tất cả các nơ-ron ở lớp tiếp theo. Điều này có nghĩa là đầu ra của mỗi nơ-ron ở lớp trước trở thành đầu vào cho mỗi nơ-ron 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 (neuron):** 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 của chúng ta có d đặc trưng thì vector đặc trưng là $\mathbf{x}_i \in \mathbb{R}^d$ hay $\mathbf{x}_i = [x_{i_1}, x_{i_2}, \dots, x_{i_d}]^T$. 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 giữa dữ 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/nhân.

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

Quá trình huấn luyện của một mạng 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 đo đặc 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,..

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:

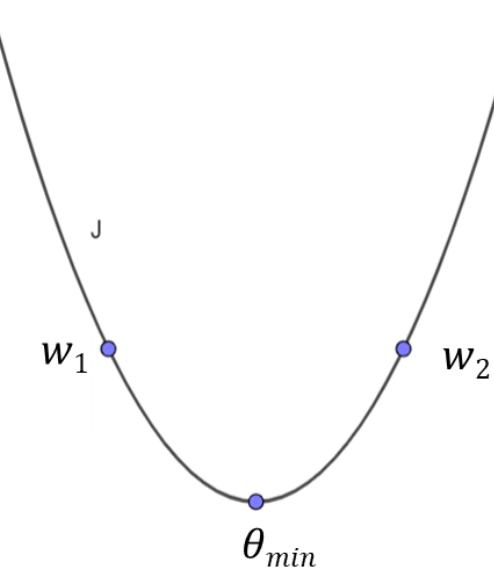
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m L(y_i, \hat{y}_i)$$

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 θ_{\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:

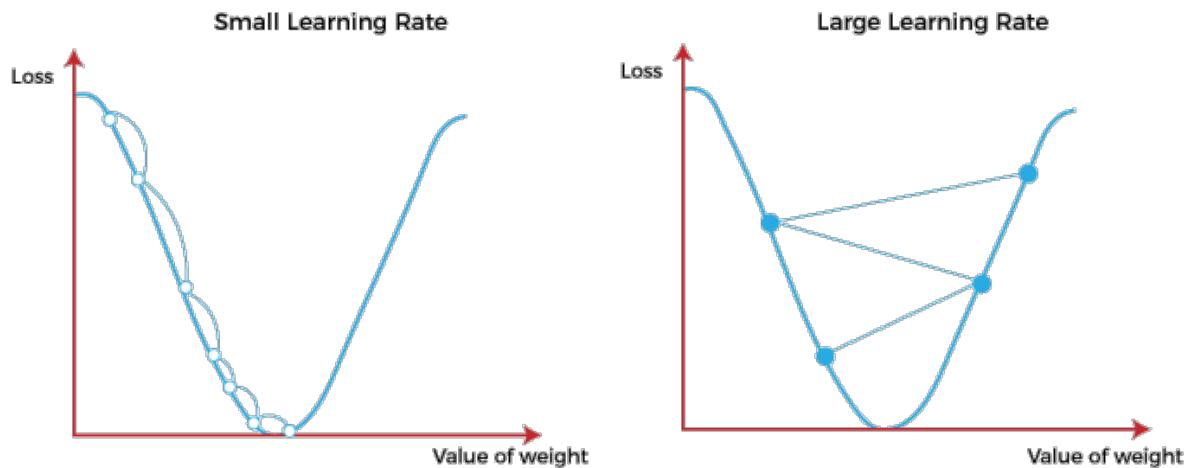
$$w_{i+1} = w_i - \eta \nabla J(w_i)$$

Vì $\nabla J(w_i)$ âm, tham số w_i sẽ tăng. Điều này giúp di chuyển w_i về phía điểm cực tiểu θ_{\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 giúp ích cho việc giảm giá trị hàm lỗi huấn luyện.

Thuật toán gradient 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 θ_{\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 tụ 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.

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:



Hình 2.4: Learning rate thấp và cao. Nguồn: Javatpoint.com[7]

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

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

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

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

- 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, chúng ta sẽ sử dụng hàm lỗi huấn luyện Cross Entropy. Trong bước 3, trước khi cập nhật tham số θ ta sẽ tính gradient của hàm lỗi này 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 [9], RMSprop [6],...

2.2 Phương pháp chống quá khớp Weight Decay và Validation

Weight Decay (Krogh và Hertz, 1992)[12], còn có tên gọi khác là L2 regularization, là một kỹ thuật không còn xa lạ, được sử dụng phổ biến để giảm thiểu overfitting.

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 tập kiểm định. 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 overfitting 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 của chúng ta 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, phương pháp này 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.

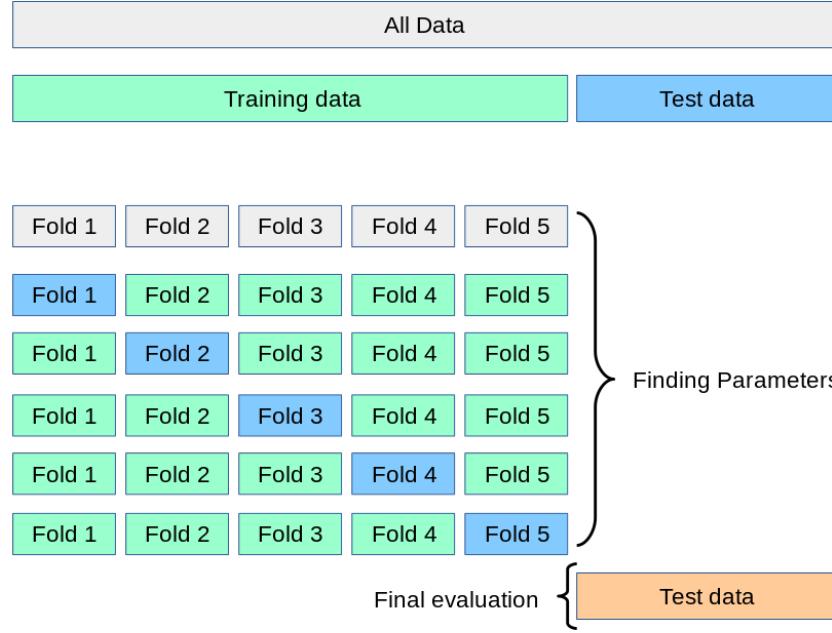
2.3.2 Nguyên lý hoạt động INCV

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 overfitting. Cross-Validation cũng sẽ dùng một phần nhỏ của dữ liệu để làm tập kiểm định và huấn luyện trên phần còn lại giống như sử dụng validation set trình bày ở mục 2.2 nhưng sẽ có thêm một số bước khác nữa phức tạp hơn. Tuy nhiên, kỹ thuật này cũng cung cấp đánh giá chính xác và toàn diện hơn về mô hình so với validation set.

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. Nguồn: scikit-learn.org[3]

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 nhiều [4] 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 nhiều 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 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 overfitting. 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, chúng ta cần mong đợi rằng 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 đúng. Tuy nhiên, nó chỉ đúng khi tỷ lệ nhiễu khá thấp và trong những trường hợp đó 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 , tỉ lệ loại bỏ r

- 1: Tập dữ liệu được chọn $S = \emptyset$, tập dữ liệu ứng viên $C = D$
- 2: **for** $i = 1, \dots, N$ **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, chúng ta có thể 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 thực 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

Trong chương này, chúng em tập trung trình bày các nội dung liên quan đến phương pháp mà chúng em chọn để giải quyết bài toán học có giám sát với dữ liệu nhiều. Đầu tiên chúng em trình bày về dạng mô hình được sử dụng trong phương pháp, cụ thể là mạng tích chập CNN (Convolutional Neural Network). Tiếp theo, chúng em trình bày ý tưởng chính của phương pháp, rồi sau đó tiếp tục đi sâu vào từng bước mà phương pháp thực hiện. Cuối cùng, chúng em có đưa ra một số đánh giá và nhận xét của chúng em trong quá trình tìm hiểu và cài đặt phương pháp này.

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 nhiều, 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

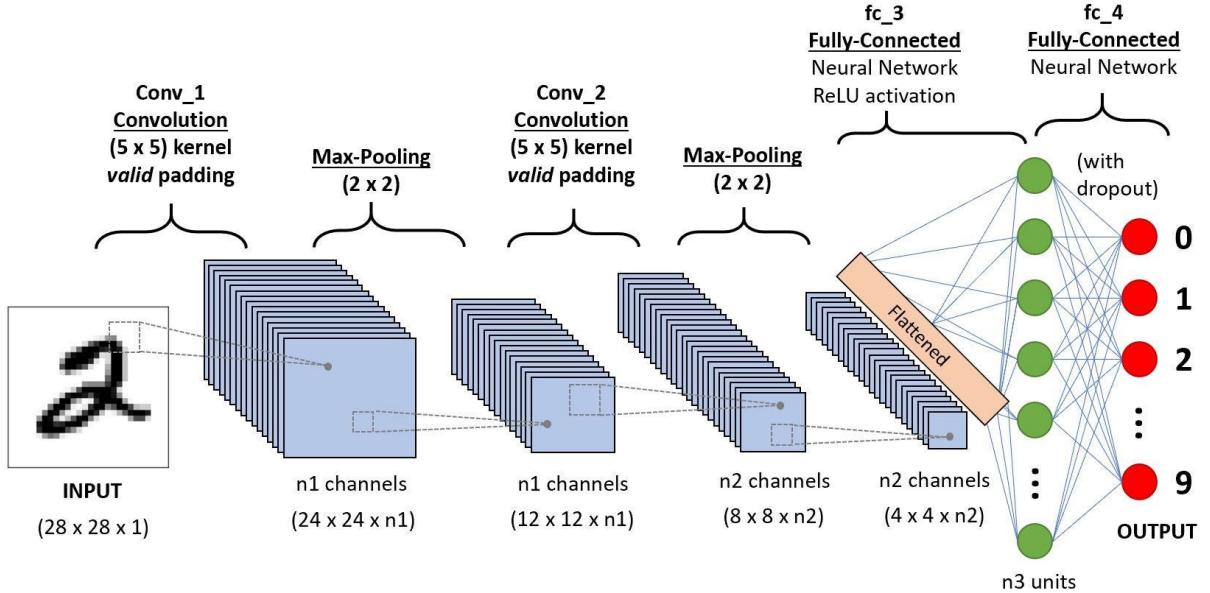
Mạng nơ-ron tích chập CNN (Convolutional neural network) là một dạng mô hình/kiến trúc được thiết kế đặc biệt để xử lý dữ liệu dạng hai chiều, có ứng dụng rộng rãi trong các bài toán thị giác máy tính như phân loại hình ảnh, phát hiện vật thể, nhận diện khuôn mặt,... Từ “convolution” trong tên của dạng mô hình này có nghĩa là tích chập, đây chính là phép toán cốt lõi để tạo nên mạng tích chập. 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 tích chập CNN được đề xuất lần đầu vào năm 1988 bởi Yann Lecun[13] 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 không thể học được mối quan hệ giữa các pixel ảnh liền kề. Ngoài ra, chi phí để huấn luyện nó cũng rất cao. Giả sử, đầu vào là một bức ảnh RGB với kích thước 256x256 và lớp ẩn đầu tiên của mô hình FCFFNN có 500 nơ-ron, chúng ta sẽ phải cần 256x256x3x500 ~ 98 triệu tham số chỉ riêng cho lớp ẩn đầu tiên, đây là một điều không tưởng. Nếu làm theo kiến trúc của mạng CNN, việc học không chỉ hiệu quả hơn mà còn nhanh hơn nhiều. Đó chính là lý do vì sao dạng mô hình/kiến trúc mạng 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 mạng nơ-ron tích chập.

3.1.2 Các thành phần cơ bản của mạng tích chập

Kiến trúc tổng thể

Mạng tích chập 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. Nguồn: medium.com[22]

Chúng ta có thể tóm tắt chức năng của các thành phần chính có mặt trong mạng tích chập 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).
- Lớp gộp - pooling layer, có chức năng chính là giảm kích thước không gian đặc trưng, từ đó giảm tham 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 tích chập là loại mạng nơ-ron sử dụng các loại lớp trên, sắp xếp chúng chồng lên nhau một thứ tự hợp lý, ngoài ra trong một số kiến

trúc cũ 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.

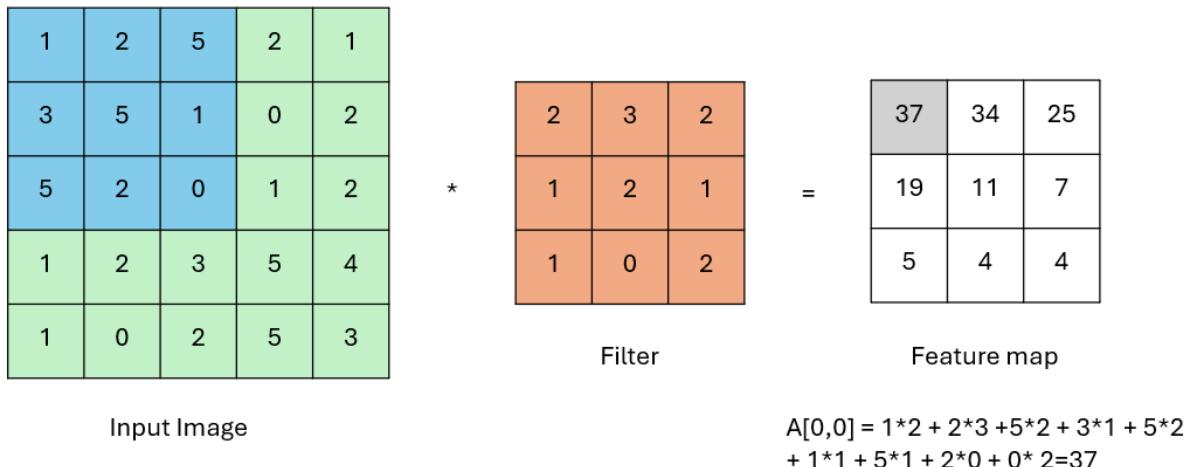
Lớp tích chập - convolution layer

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 dạng mô hình/kiến trúc CNN. Công dụng chính của lớp này cũng chính 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 3×3 hoặc 5×5 . 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 bản đồ đặc trưng 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 6×6 , bộ lọc 3×3 , 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 không xung quanh đường biên của ma trận trước khi áp dụng phép tích chập. Một số kiểu thêm đường viền phổ biến được người ta hay 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 chất 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 loại hàm thường được sử dụng đại diện cho hai loại lớp gộp:

- **Max pooling:** khi bộ lọc di chuyển qua đầu vào (thường là feature map của lớp tích chập), 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 thông tin nhưng bằng cách giảm độ phức tạp của mô hình, nó sẽ giúp hạn chế việc quá khớp (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 neuron 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 (cho phân loại nhiều lớp) hoặc sigmoid (cho phân loại nhị phân) để sản sinh ra xác suất dự đoán cuối cùng. Đây là cách mà mạng neural 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 tích chập - 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[13]

- AlexNet[11]
- VGGNet[18]
- ResNet[5]
- EfficientNet[19]

Trong đó, mô hình ResNet-32 được chúng em sử dụng để thực hiện thí nghiệm với CRUST, thông tin này sẽ được trình bày rõ hơn ở chương 4.

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. 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[23].

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 quá khớp (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 cao, 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”[16]. 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. Tác giả tìm cụm này bằng cách quy về giải quyết bài toán k-medoids. 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.

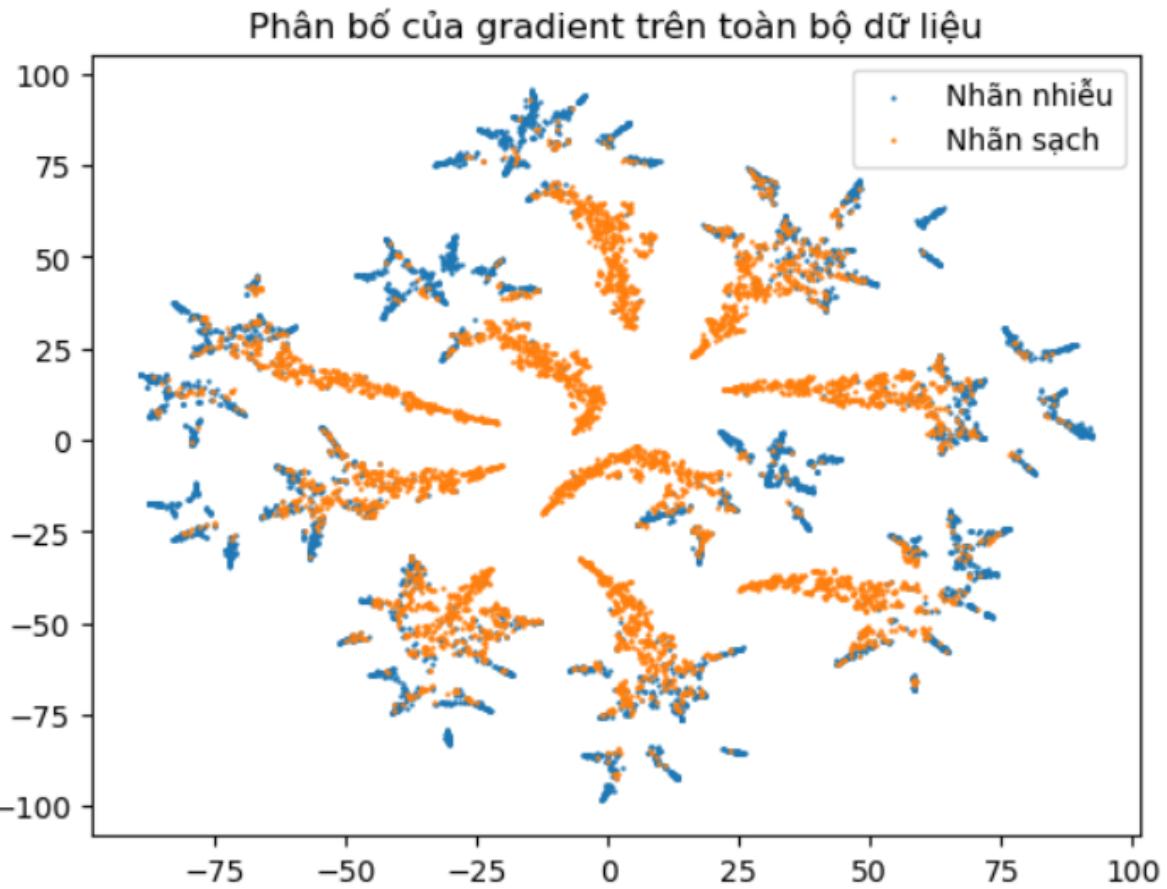
Hình 3.3 thể hiện phân bố của gradient trên tập dữ liệu CIFAR với tỉ lệ nhiễu 50%. Mặc dù đã sử dụng t-SNE[20] để giảm chiều gradient về còn 2, sự phân bố có thể không hoàn toàn chính xác, nhưng nó cũng có thể phản ánh được một phần. Chúng ta có thể dễ dàng nhìn thấy các điểm dữ liệu sạch phân bố gần nhau và các điểm dữ liệu nhiễu thường sẽ có sự phân bố tách biệt với các điểm dữ liệu sạch.

Phương pháp CRUST tận dụng hai quan sát trên, tìm kiếm coreset bằng cách xác định các điểm dữ liệu sạch nằm ở trung tâm nhất. Các điểm này có thể tìm được thông qua việc giải quyết bài toán k-medoids sau: tìm cụm dữ liệu sao cho mỗi điểm trong cụm đó có tổng khoảng cách đến tất cả các điểm còn lại trong dữ liệu là nhỏ nhất.

3.2.3 Các bước thực hiện

Phương pháp CRUST là một kỹ thuật chọn lọc dữ liệu. Tương tự như các phương pháp chọn lọc dữ liệu khác, CRUST bao gồm việc lựa chọn dữ liệu sạch và huấn luyện mô hình trên tập dữ liệu đã được chọn. Tuy nhiên, khác với những phương pháp truyền thống thường chọn dữ liệu sạch dựa trên việc đánh giá độ mất mát (loss) thấp trong quá trình huấn luyện, CRUST sử dụng việc quan sát gradient để chọn dữ liệu. Dưới đây, chúng em sẽ trình bày các bước cụ thể của CRUST để chọn dữ liệu sạch và huấn luyện mô hình trên tập dữ liệu đó. Các bước của thuật toán như sau

Với mỗi lần lặp:



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.

- Chọn coresnet theo từng lớp của nhãn:
 - Chọn ra tập con của dữ liệu có nhãn c là lớp đang xét.
 - Sử dụng thuật toán tham lam để giải bài toán k-medoids tìm coresnet trong tập con có nhãn c .
- Gộp các coresnet vừa tìm được.
- Sử dụng coresnet để huấn luyện mô hình.

Trước khi đi sâu tìm hiểu rõ hơn phương pháp CRUST, chúng em sẽ giải thích thêm một số chi tiết nhỏ mà phương pháp này đã sử dụng. Trước hết, CRUST không chọn coresnet trên toàn bộ dữ liệu mà chọn theo từng lớp, tại sao lại làm như vậy? Giải thích cho điều này là do phân bố gradient của dữ liệu sẽ ưu tiên gom cụm theo lớp trước, sau đó trong cùng một lớp

mới có gom cụm theo nhãn nhiễu và nhãn sạch. Vì vậy, nếu chọn coreset trên toàn bộ dữ liệu sẽ khó khăn hơn so với lần lượt từng nhãn và có thể chọn trùng các cụm nhiễu. Trong chương 4, chúng em làm rõ hơn vấn đề này qua các thí nghiệm quan sát gradient của dữ liệu.

Như những gì đã nói trên, việc chọn coreset theo từng lớp sẽ giúp lọc ra chính xác các dữ liệu sạch hơn. Như vậy, trước hết chúng ta sẽ lọc ra những mẫu có nhãn là c trước, sau đó mới chọn coreset trên những mẫu này. Nhận thấy rằng nhãn chúng ta quan sát được chỉ là nhãn nhiễu, tác giả đưa ra cải tiến mới rằng thay vì chọn những mẫu có nhãn (nhiễu) là c , hãy chọn những mẫu được dự đoán là thuộc lớp c (hay nhãn dự đoán bằng c). Trong bài báo gốc, tác giả không có giải thích thêm tại sao lại phải sử dụng nhãn dự đoán, nhưng họ đã thực hiện các thí nghiệm khác nhau sử dụng nhãn dự đoán và nhãn nhiễu sau đó dùng kết quả này để chứng minh rằng việc sử dụng nhãn dự đoán để phân loại sẽ tốt hơn. Tuy nhiên sau khi chúng em thực hiện lại nhiều thí nghiệm, kể cả sử dụng code gốc được public ở bài báo, kết quả lại cho thấy rằng việc sử dụng nhãn dự đoán sẽ làm giảm hiệu suất trong hầu hết các trường hợp, nội dung cụ thể trình bày ở chương 4.

Tổng kết, phương pháp CRUST sẽ chọn ra coreset từ dữ liệu sau đó 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 tập con bằng thuật toán k-medoids

Khái niệm medoids

Medoids là một khái niệm trong lĩnh vực phân cụm dữ liệu (clustering). Một medoid là một điểm trong một cụm dữ liệu mà khoảng cách tổng thể đến tất cả các điểm khác trong cụm là nhỏ nhất. Medoids là những điểm đại diện trong một tập dữ liệu.

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 .

- 1: **for** $\tau = 1, \dots, T$ **do**
 - 2: $S^\tau = \emptyset$.
 - 3: **for** $c \in \{1, \dots, C\}$ **do**
 - 4: Chọn ra tập U_c^τ từ tập \mathcal{D} là các phần tử có nhãn thuộc lớp c .
 - 5: Tính khoảng cách gradient d_{ij} giữa các điểm dữ liệu trong U_c^τ .
 - 6: Sử dụng thuật toán k-medoids để tìm ra tập coreset S_c^τ theo lớp c .
 - 7: **end for**
 - 8: Gộp các tập coreset từng lớp thành tập coreset để huấn luyện S^τ .
 - 9: Huấn luyện mô hình trên coreset S^τ và cập nhật tham số W^T .
 - 10: **end for**
-

Bài toán k-medoids để tìm tập con

Tận dụng quan sát các điểm dữ liệu có nhãn sạch thường có xu hướng gom cụm lại với nhau trong không gian gradient, bài toán k-medoids thực hiện chọn k medoids từ tập dữ liệu gốc để tạo thành một coreset gồm các điểm dữ liệu có nhãn sạch mang các đặc trưng quan trọng của tập dữ liệu ban đầu.

Cụ thể bài toán được phát biểu như sau:

$$S^*(\mathbf{W}) \in \arg \min_{S \subseteq V} \sum_{i \in V} \min_{j \in S} d_{ij}(\mathbf{W}) \quad \text{s.t.} \quad |S| \leq k, \quad (3.1)$$

Trong đó:

- S là tập coreset được chọn
- V là tập hợp tất cả các điểm dữ liệu.
- k là số điểm cần tìm cho coresnet
- d_{ij} là khoảng cách gradient giữa 2 điểm dữ liệu i và j

Mục tiêu của bài toán là tìm tập coreset S gồm k điểm từ tập dữ liệu V sao cho tổng khoảng cách từ các điểm dữ liệu khác đến medoid gần nhất

của nó là nhỏ nhất. Khoảng cách này được tính bằng khoảng cách gradient giữa các điểm dữ liệu, cụ thể là:

$$d_{ij}(\mathbf{W}) = \|\nabla \mathcal{L}(\mathbf{W}, x_i) - \nabla \mathcal{L}(\mathbf{W}, x_j)\|_2, \quad (3.2)$$

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

Gradient trong việc chọn coresnet

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[8]. 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 giữa hai điểm dữ liệu x_i và x_j được tính xấp xỉ như sau:

$$d_{ij} \approx \|\Sigma'_L(z_i^L) \nabla_i^L \mathcal{L} - \Sigma'_L(z_j^L) \nabla_j^L \mathcal{L}\|_2 \quad (3.3)$$

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.

Giải pháp cho bài toán k-medoids

Bài toán k-medoids là một bài toán NP-khó[15], tức là không có giải pháp tối ưu nào có thể tìm được trong thời gian đa thức cho mọi trường hợp. Tuy nhiên, có nhiều phương pháp xấp xỉ có thể cung cấp giải pháp gần tối ưu một cách hiệu quả. Một trong số đó là sử dụng tính chất submodular của hàm mục tiêu và áp dụng thuật toán tham lam (greedy)[21].

Hàm submodular và thuật toán tham lam

Một hàm $F : 2^V \rightarrow \mathbb{R}^+$ được gọi là submodular nếu nó thỏa mãn tính chất giảm dần của lợi ích biên: thêm một phần tử vào một tập hợp nhỏ sẽ tạo ra mức tăng giá trị lớn hơn so với khi thêm phần tử đó vào một tập hợp lớn hơn. Cụ thể, hàm F là submodular nếu với mọi $S \subseteq T \subseteq V$ và mọi $e \in V \setminus T$, ta có:

$$F(S \cup \{e\}) - F(S) \geq F(T \cup \{e\}) - F(T), \quad (3.4)$$

Bài toán k-medoids có thể được biểu diễn lại dưới dạng bài toán tối đa hóa hàm submodular:

$$S^*(\mathbf{W}) \in \arg \max_{S \subseteq V, |S| \leq k} F(S, \mathbf{W}), \quad (3.5)$$

với

$$F(S, \mathbf{W}) = \sum_{i \in V} \max_{j \in S} (d_0 - d_{ij}(\mathbf{W})), \quad (3.6)$$

trong đó d_0 là một hằng số lớn hơn khoảng cách lớn nhất giữa các khoảng cách gradient của các điểm dữ liệu.

Bằng cách biến đổi bài toán k-medoids thành bài toán tối đa hóa hàm submodular, chúng ta có thể áp dụng thuật toán tham lam để tìm kiếm giải pháp. Thuật toán tham lam hoạt động bằng cách chọn các điểm dữ liệu từng bước sao cho mỗi bước thêm một điểm mới vào tập hiện tại sao cho giá trị hàm submodular tăng nhiều nhất.

Thuật toán tham lam để chọn coresnet

Thuật toán tham lam để chọn coresnet từ tập dữ liệu gốc có thể được mô tả như sau:

- Khởi tạo:
 - Bắt đầu với tập rỗng $S = \emptyset$.
 - Xác định số điểm cần chọn k .
- Tại mỗi vòng lặp t :

- Tìm điểm $e \in V$ sao cho $F(e|S_t) = F(S_t \cup \{e\}) - F(S_t)$ đạt giá trị lớn nhất.
- Thêm điểm e vào tập S .
- Trả về tập coreset S .

Hàm submodular có tính chất giảm dần của lợi ích biên, giúp thuật toán tham lam hoạt động hiệu quả. Tuy nhiên, để tối ưu hóa hơn nữa, chúng ta có thể sử dụng thuật toán tham lam lười (lazy greedy)[14]. Thuật toán này được thiết kế để cải thiện hiệu quả của thuật toán tham lam truyền thống. Thay vì tính toán lại giá trị hàm mục tiêu cho mọi điểm ở mỗi bước, thuật toán tham lam lười sử dụng một max heap để lưu trữ và truy xuất các giá trị lợi ích biên lớn nhất một cách nhanh chóng. Thuật toán này giúp giảm bớt số lần tính toán không cần thiết, từ đó tăng tốc độ chọn lựa tập con.

3.2.5 Nhận xét và đánh giá

Hiệu quả của phương pháp

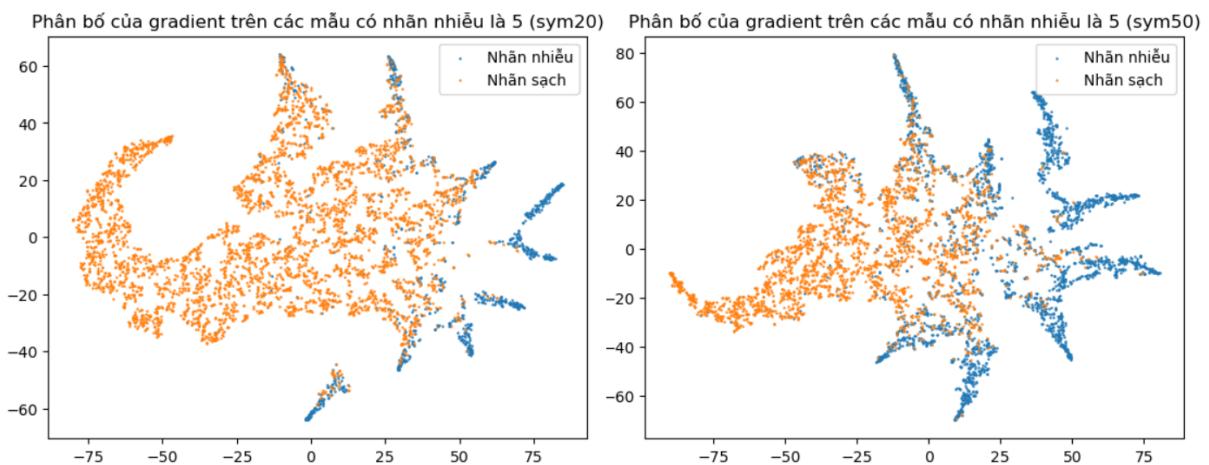
Phương pháp CRUST là một hướng nghiên cứu mới mẻ trong lĩnh vực học sâu khi đối phó với vấn đề dữ liệu nhiễu. Ý tưởng sử dụng các coreset để chọn lọc dữ liệu sạch và sự quan sát phân bố của gradient từ tập dữ liệu nhiễu là một cách tiếp cận sáng tạo và có tiềm năng lớn cho việc tiếp tục cải tiến, nâng cao hiệu suất.

Phương pháp này cũng đã được kiểm chứng và chứng minh hiệu suất thông qua nhiều thí nghiệm thực tế. Các thí nghiệm này cho thấy rằng CRUST có khả năng vượt trội trong việc giảm thiểu tác động của nhiễu nhiễu và cải thiện độ chính xác của mô hình trên các tập kiểm thử. Phần này sẽ được thể hiện chi tiết hơn ở chương 4.

Ngoài ra, quy trình xác định các điểm medoid trong không gian gradient là một bước quan trọng và có ý nghĩa trong phương pháp CRUST. Các medoid được lựa chọn sao cho chúng đại diện tốt nhất cho các điểm dữ liệu sạch và có tính đại diện cao. Việc sử dụng cách làm này không chỉ giúp tìm ra cá điểm dữ liệu sạch, đồng thời đảm bảo rằng các điểm này vẫn giữ được các đặc điểm quan trọng của toàn bộ tập dữ liệu.

Nhận định về sự phân bố của gradient

Mặc dù phương pháp CRUST có nhiều điểm tốt, nhưng với một cách tiếp cận mới mẻ này, vẫn còn rất nhiều điều khiến chúng ta nghi vấn về tính hiệu quả của phương pháp. Điều đầu tiên chính là: Liệu nhận định về gradient của các điểm dữ liệu sạch sẽ gom cụm và gradient của các điểm dữ liệu nhiễu sẽ phân tán trong không gian gradient có đúng? Và đúng trong trường hợp nào?



Hình 3.4: Phân bố của gradient của các mẫu có nhãn nhiễu là 5 sau khi sử dụng t-SNE để giảm chiều trên tập dữ liệu CIFAR-10 khi huấn luyện với mạng nơ-ron. Hình bên trái tập dữ liệu được điều chỉnh với tỷ lệ nhiễu 20%, hình bên phải được điều chỉnh với tỷ lệ nhiễu 50%

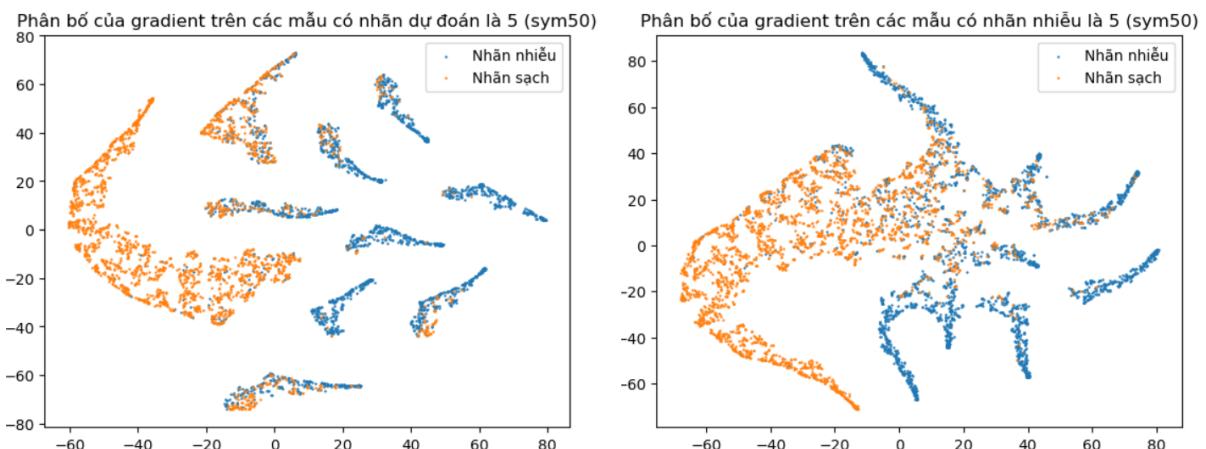
Hình 3.4 thể hiện sự phân bố gradient trên 1 lớp cụ thể trên tập dữ liệu CIFAR-10 khi được điều chỉnh ở hai tỉ lệ nhiễu khác nhau. Dễ thấy rằng khi tỉ lệ nhiễu tăng từ 20% lên 50%, lúc đó các các điểm dữ liệu có nhãn nhiễu cũng bắt đầu có xu hướng gom cụm. Ảnh hưởng cụ thể của việc dữ liệu nhiễu cũng gom cụm sẽ được thể hiện chi tiết hơn ở thí nghiệm mở rộng của chương 4. Qua đây chúng ta có thể thấy rằng, khi tỉ lệ nhiễu cao, nhận định này không còn hoàn toàn đúng, và việc tìm ra điểm dữ liệu sạch càng trở nên khó khăn hơn, dẫn đến hiệu suất mô hình sẽ giảm đáng kể.

Sử dụng nhãn nhiễu và nhãn dự đoán để chọn coreset

Mặc dù gradient gom cụm theo lớp trước, thế nên việc tìm coreset trong từng lớp thì sẽ mang lại hiệu quả tốt hơn. Nhưng việc sử dụng nhãn nhiễu

hay nhãn dự đoán làm mục tiêu để chọn coreset sẽ tạo ra một kết quả khác nhau.

Sử dụng nhãn nhiễu để chọn coreset tức là chúng ta sẽ sử dụng phương pháp CRUST để tìm coreset trong từng lớp trên những mẫu có cùng nhãn mà chúng ta nhìn thấy được (nhãn nhiễu) là c . Tập dữ liệu này sẽ bao gồm những mẫu dữ liệu có nhãn sạch là c và những mẫu nhãn nhiễu khác nhưng được gán nhãn là c . Những mẫu nhiễu này đều hoàn toàn ngẫu nhiên, nhãn thực sự của nó có thể thuộc bất cứ lớp nào. Còn sử dụng nhãn dự đoán thì chúng ta sẽ thực hiện tìm coreset trên những mẫu mà nó cùng được dự đoán sẽ thuộc lớp c khi đưa vào mô hình. Trong tập dữ liệu này thì bao gồm những mẫu có nhãn sạch là c và những mẫu nhãn nhiễu khác nhưng vẫn được mô hình dự đoán là thuộc lớp c . Những mẫu nhãn nhiễu này rất có thể nhãn thực sự của nó thuộc lớp c .



Hình 3.5: Phân bố của gradient của các mẫu có nhãn nhiễu là 5 và nhãn dự đoán là 5 sau khi sử dụng t-SNE để giảm chiều. Hình bên trái là phân bố của gradient của các mẫu dự liệu được dự đoán có nhãn 5, hình bên phải là phân bố gradient của các mẫu có nhãn nhiễu là 5

Khi chọn coreset trong từng lớp theo các mẫu có cùng nhãn dự đoán thì phân bố của gradient sẽ rõ ràng hơn. Hình 3.5 thể hiện điều này. Về mặt ý tưởng, chúng ta thấy việc phân bố như vậy sẽ giúp việc chọn coreset qua bài toán k-medoids dễ dàng hơn và chính xác hơn. Nhưng qua các thí nghiệm mà chúng em đã tiến hành thì lại cho kết quả khác. Nhận định này mà tác giả đưa ra không hoàn toàn chính xác, nó có thể sẽ đúng trong trường hợp nào đó, nhưng không phải lúc nào cũng đúng. Chi tiết sẽ được

làm rõ hơn ở phần thí nghiệm trong chương 4.

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 có một số khác biệt giữa kết quả của phương pháp CRUST mà chúng em cài đặt với kết quả được công bố trong bài báo gốc. Do đó, ở thí nghiệm 2 - thí nghiệm mở rộng ngoài bài báo gốc, chúng em tiến hành phân tích tìm nguyên nhân dẫn đến các kết quả khác biệt này và tiến hành cải thiện kết quả của phương pháp CRUST mà chúng em cài đặt ở thí nghiệm 1.

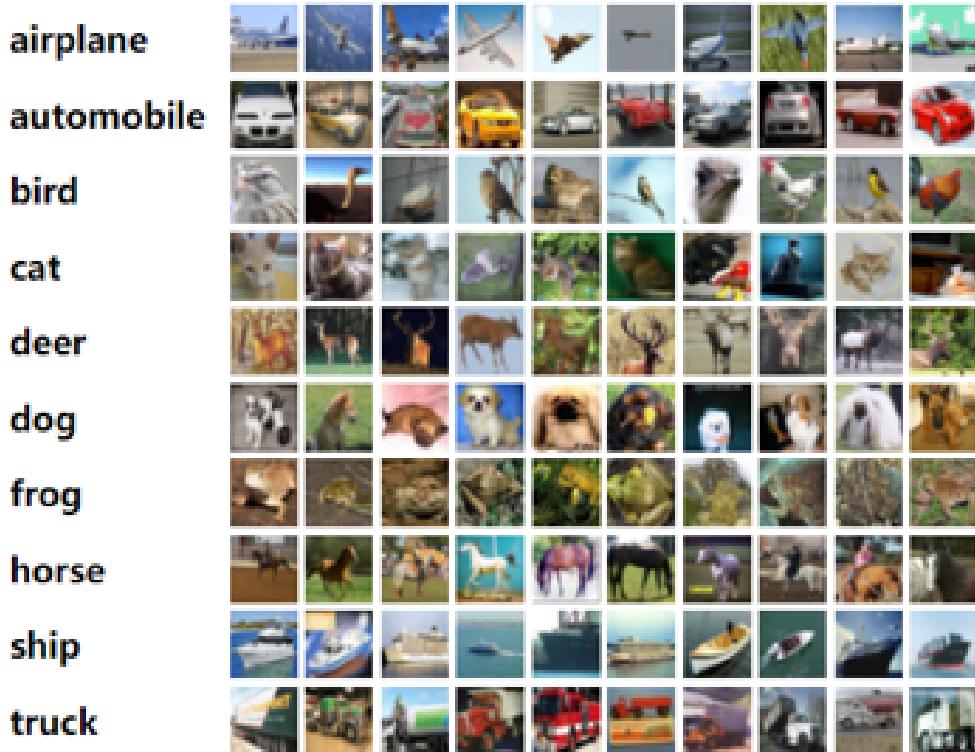
4.1 Thiết lập thí nghiệm

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 60,000 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

50,000 ảnh cho quá trình huấn luyện và 10,000 ảnh còn lại để cho quá trình kiểm tra. Cả 2 bộ dữ liệu này đều có nhãn sạch. Chính vì vậy, chúng ta có thể dễ dàng điều chỉnh nhiều để có một bộ dữ liệu với tỷ lệ nhiều như mong muốn.



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

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. Các siêu tham số của mô hình khác nhau với các thí nghiệm khác nhau, chúng em sẽ trình bày chi tiết ở từng thí nghiệm.

Trước khi đưa dữ liệu vào quá trình huấn luyện, chúng em thực hiện một số bước xử lý ảnh trước đó[5]. 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 32×32 từ ảnh gốc. Sau đó thực hiện lật ngẫu nhiên ảnh theo chiều ngang với xác suất 0.5. Việc áp dụng 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[5] 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, 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. Với mỗi thí nghiệm, chúng em thực hiện 5 lần chạy khác nhau và lấy kết quả trung bình của cả 5 lần.

Mô hình mạng nơ-ron ResNet-32

ResNet-32[5] 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 (32×32), 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 nhiều, 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

Cài đặt

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. Thế nên chúng em sử dụng phiên bản nhiễu của tập dữ liệu này[10]. Cụ thể chúng em thực hiện tạo dữ liệu nhiễu 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. Cụ thể trong thí nghiệm, 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.

Trước hết, chúng em thực hiện thí nghiệm phương pháp CRUST trong 2 trường hợp: sử dụng *nhãn dự đoán* và *nhãn nhiễu* như đã đề cập trong chương 3. 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.

Kết quả

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. Phần phương pháp có tên được in đậm là phần mà chúng em lấy kết quả từ bài báo để cho việc

so sánh.

Dataset	CIFAR-10				CIFAR-100		
Noise Type	Sym		Asym	Sym	Sym	Asym	
Noise Ratio	20	50	80	40	20	50	40
INCV (Paper)	89.7	84.8	52.3	86.0	60.2	53.1	50.7
CRUST nhān nhiều (Paper)	90.5	85.2					
CRUST nhān dự đoán (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 nhān nhiều	85.4	86.5	36.5	80.6	63.9	54.5	58.3
CRUST nhān dự đoán	84.1	84.3	24.2	67.1	60.1	50.1	44.3

Bảng 4.1: So sánh kết quả với bài báo gốc

Theo như kết quả có được ở Bảng 4.1, chúng ta có thể thấy rằng, ý tưởng dựa trên sự phân bố của gradient để xác định các mẫu sạch mẫu nhiều là một điều hoàn toàn khả thi. Phương pháp CRUST dựa trên ý tưởng này cho hiệu suất tốt trong nhiều trường hợp khác nhau. Kết quả của phương pháp CRUST (phiên bản tốt nhất) mà chúng em cài đặt đạt được hiệu suất tương đương với phần do tác giả cài đặt trong nhiều trường hợp. Tuy nhiên vẫn có một số thí nghiệm cho ra kết quả khác. Về phần phương pháp INCV, đây vẫn là một phương pháp huấn luyện mạng nơ-ron với dữ liệu nhiều 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.

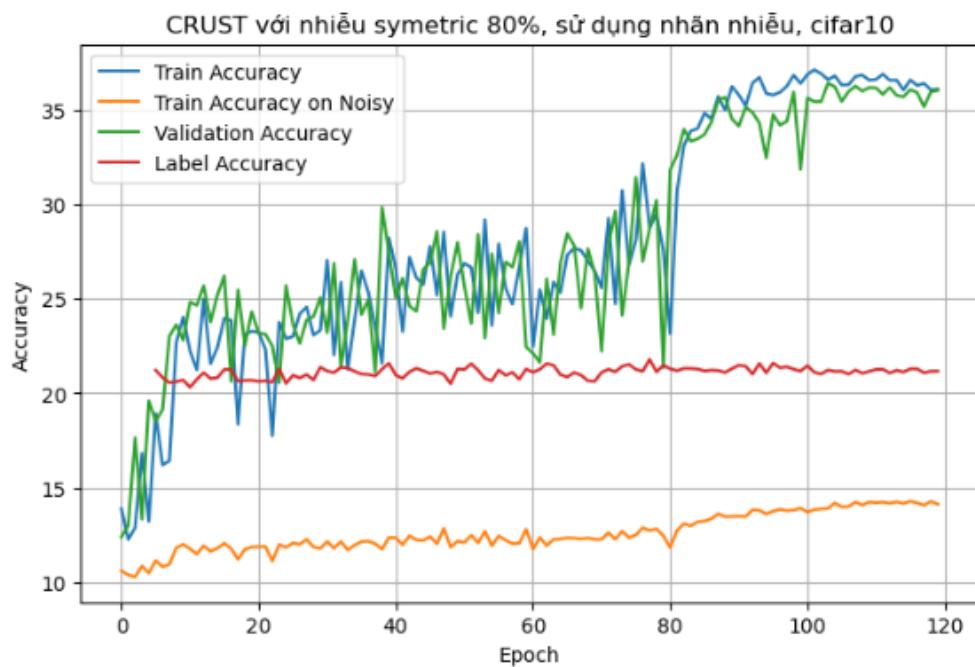
So sánh kết quả với bài báo gốc

Sau đây là các khác biệt giữa kết quả thí nghiệm so với kết quả ở bài báo gốc:

- Sử dụng nhān dự đoán cho kết quả thấp hơn sử dụng nhān
nhiều:** Theo như thí nghiệm mà chúng em đã thực hiện, phương
pháp CRUST sử dụng nhān dự đoán để tìm coreset cho kết quả thấp
hơn so với phương pháp CRUST khi sử dụng nhān nhiều trong tất

cả trường hợp. Điều này trái ngược với điều mà tác giả đã nhắc đến trong bài báo.

- **Kết quả sym 20 thấp hơn sym 50:** Điều tiếp theo mà chúng ta có thể nhận thấy từ kết quả thí nghiệm là kết quả ở sym 20 thấp hơn kết quả ở sym 50. Mặc dù cùng một kiểu nhiễu giống nhau, nhưng với tỉ lệ nhiễu thấp hơn là 20 nhưng kết quả lại không tốt hơn tỉ lệ nhiễu cao hơn là 50.
- **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 4.2 cho thấy Label Accuracy (tỉ lệ nhãn chính xác trong coresnet) rất thấp, hầu như là chọn một cách ngẫu nhiên. Chúng ta 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.



Hình 4.2: Biểu đồ độ chính xác của mô hình khi huấn luyện trên tập CIFAR với sym 80

- Độ chính xác trên tập validation thấp hơn INCV ở một số thí nghiệm trên CIFAR-10 nhưng cao hơn trên CIFAR-100: Trên CIFAR-10, CRUST có độ chính xác trên tập validation thấp hơn INCV ở sym 20, nhưng trên CIFAR-100 lại cao hơn. Tương tự, với asym 40 cũng xảy ra trường hợp này.

Để làm rõ hơn nguyên nhân và các giải thích cho sự khác biệt trên, chúng em đã tiến hành thêm một số thí nghiệm để kiểm tra các điều này và được trình bày ở trong phần tiếp theo

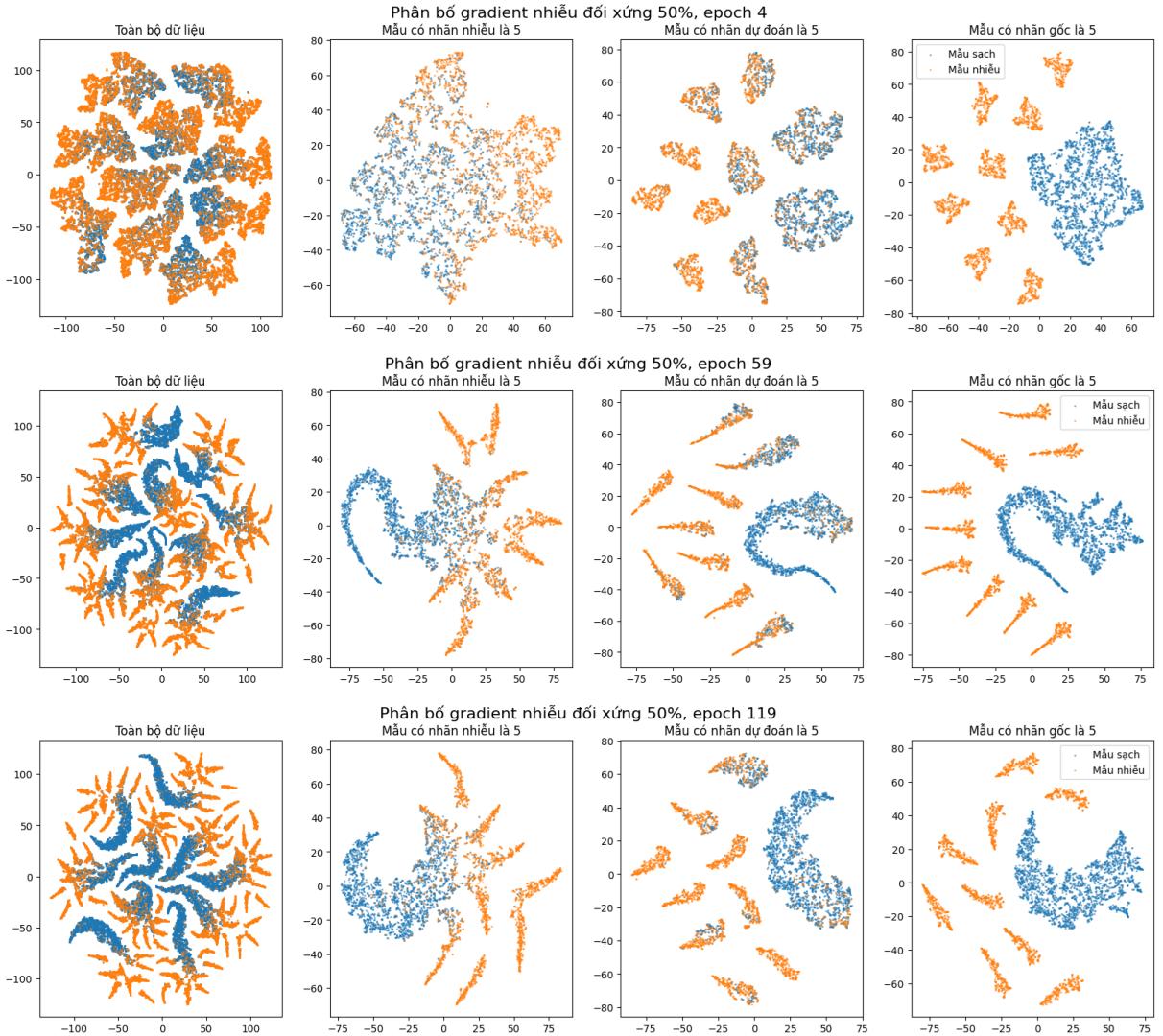
4.3 Thí nghiệm 2: phân tích và cải thiện kết quả của phương pháp CRUST ở thí nghiệm 1

4.3.1 Thí nghiệm về sự phân bố của gradient

Phương pháp CRUST là một phương pháp huấn luyện trên dữ liệu nhiễu theo hướng tiếp cận mới mẻ đó là dựa trên các quan sát về sự phân bố của gradient. Phương pháp này chọn dữ liệu sạch dựa trên quan sát “các điểm dữ liệu sạch thường gom cụm với nhau trên không gian gradient”. Trong thí nghiệm, này, chúng em kiểm định nhận định trên bằng cách thực hiện huấn luyện mô hình trên các mức độ nhiễu khác nhau, sau đó trực quan phân bố gradient của các điểm dữ liệu ở epoch 4, 59 và 119. Việc trực quan ở các epoch trên sẽ biết được phân bố gradient của đầu, giữa cuối quá trình huấn luyện và hiểu hơn lý do tại sao tác giả lại chọn epoch 4 làm epoch mặc định để bắt đầu sử dụng CRUST (trong phần cài đặt CRUST được bật lên sau khi huấn luyện mô hình bình thường trong một số epoch đầu).

Với mỗi thí nghiệm chúng em trực quan để xem phân bố gradient trên toàn bộ dữ liệu và khi lọc dữ liệu theo lớp c sử dụng nhãn nhiễu, nhãn dự đoán, nhãn thật. Để tiện hơn cho trực quan, chúng em sử dụng t-SNE để giảm dữ liệu còn 2 chiều, và chỉ lấy 10000 điểm đầu tiên của dữ liệu (dữ liệu đã được trộn ngẫu nhiên trước khi huấn luyện). Ngoài ra trong thí nghiệm này mục đích chính là để quan sát phân bố của gradient nên chúng em cũng không sử dụng CRUST.

Phân bố gradient theo số epoch



Hình 4.3: Phân bố gradient của nhiễu đối xứng 50%

Hình 4.3 trực quan phân bố gradient của nhiễu đối xứng 50%. Chúng ta có thể thấy rằng gradient của các điểm dữ liệu sạch sẽ gom cụm với nhau, dữ liệu nhiễu cũng sẽ gom cụm với nhau. Khi huấn luyện càng nhiều epoch, mô hình bắt đầu có thể phân biệt dữ liệu sạch và nhiễu một cách rõ ràng hơn (gradient của các cụm tách biệt hơn). Điều này cho thấy cách chọn coreset của phương pháp CRUST là đúng. Mặt khác, việc tác giả chọn bật CRUST sau 5 epoch là có cơ sở. Mặc dù bật CRUST càng sớm là càng tốt, nhưng bật sớm có thể làm tốc độ huấn luyện chậm hơn. Hơn nữa những epoch đầu gradient vẫn còn hỗn loạn, đến epoch 4 mới bắt đầu có mầm mống của sự phân cụm một cách rõ ràng, vì vậy đây mới là

thời điểm phù hợp để bật CRUST.

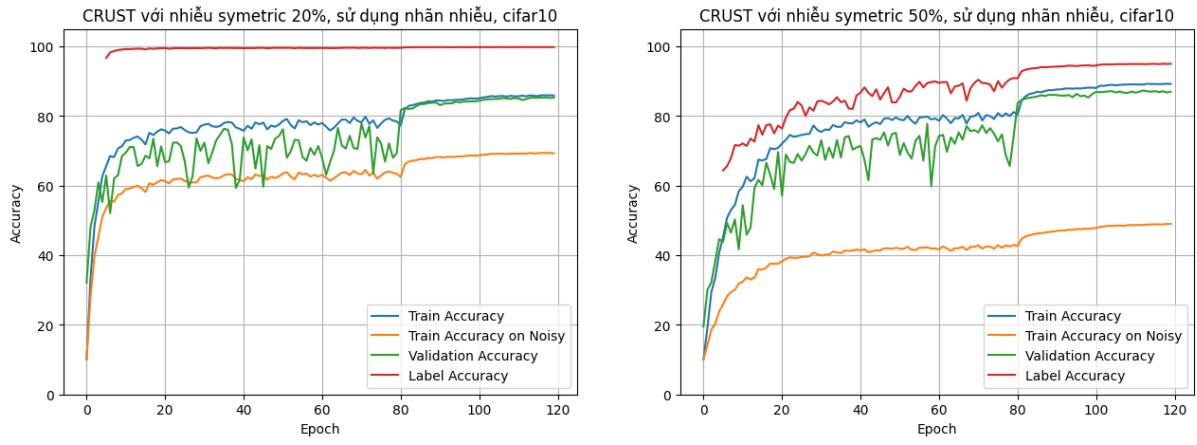
Phân bố gradient khi sử dụng các loại nhãn khác nhau để gom cụm

Phương pháp CRUST sẽ chọn coreset theo từng lớp rồi mới gộp lại. Lý do làm như vậy là do gradient của các điểm dữ liệu sẽ ưu tiên gom cụm theo lớp trước rồi mới thực hiện gom cụm theo dữ liệu nhiễu và sạch. Hình 4.3 cho thấy phân bố gradient khi xét trên toàn bộ dữ liệu thì sẽ chia thành nhiều cụm lớn, số cụm này đúng bằng số lớp. Vậy nên nếu như chúng ta chia tách ra từng lớp để chọn coreset sẽ chính xác hơn nhiều.

Việc tách ra từng lớp để chọn coreset có thể được thực hiện bằng cách chọn ra những mẫu có những nhãn quan sát được (nhãn nhiễu) là c hoặc có nhãn dự đoán là c. Hình 4.3 trực quan phân bố gradient của các loại nhãn khác nhau này. Theo quan sát từ hình trên, nếu chúng ta có thể sử dụng nhãn thật để tách lớp là tốt nhất, gradient sẽ phân cụm rõ ràng. Tuy nhiên nhãn thật sẽ không quan sát được, mục đích sử dụng của nó chỉ là để kiểm chứng nhận định về gom cụm. Tiếp theo, gradient của nhãn dự đoán phân cụm rõ ràng hơn gradient của nhãn nhiễu. Vì vậy, ý tưởng sử dụng nhãn dự đoán để tách lớp là hợp lý. Tuy nhiên, kết quả thực nghiệm lại cho thấy rằng sử dụng nhãn nhiễu sẽ cho kết quả tốt hơn trong tất cả trường hợp, phần này sẽ giải thích rõ ràng hơn ở mục 4.3.3.

4.3.2 Phân tích tìm nguyên nhân kết quả của nhiễu sym 20 và nhiễu sym 50 trên CIFAR-10

Hình 4.4 so sánh kết quả huấn luyện của hai loại nhiễu đối xứng 20% và nhiễu đối xứng 50%. Trong đó Label Accuracy là độ chính xác của nhãn (hay là tỷ lệ dữ liệu sạch), Train Accuracy là độ chính xác trên tập huấn luyện (sử dụng nhãn thật để đo), Train Accuracy on Noisy là độ chính xác trên tập huấn luyện (sử dụng nhãn nhiễu để đo), Validation Accuracy là độ chính xác trên tập validation (sử dụng nhãn thật để đo).

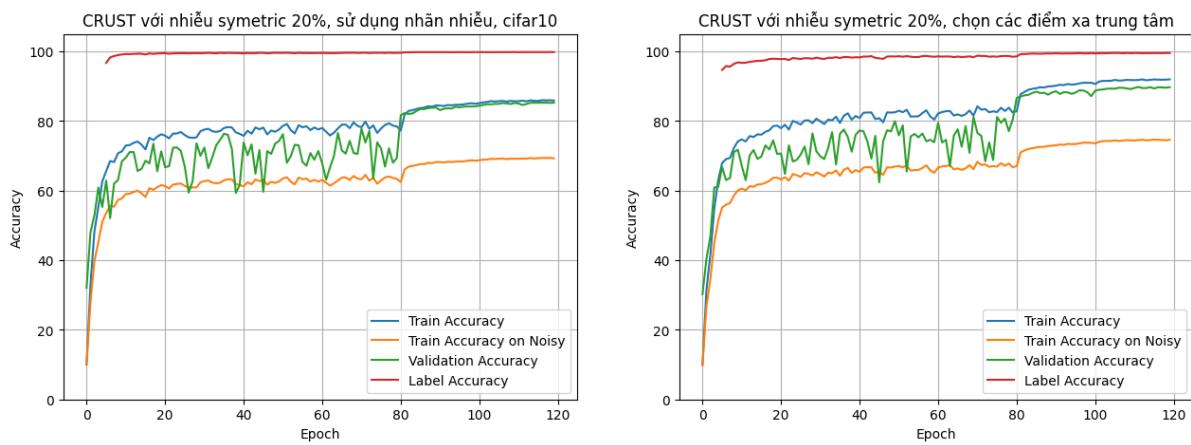


Hình 4.4: So sánh kết quả huấn luyện nhiễu đố xứng 20% và 50% trên CIFAR-10

Phân tích tìm nguyên nhân

Có thể thấy rằng tỷ lệ nhãn sạch của nhiễu symetric 20% gần như là 100%, điều này chứng minh rằng phương pháp CRUST làm khá tốt trong việc chọn dữ liệu sạch. Tuy nhiên ở đây độ chính xác trên tập validation chỉ đạt 85.4%, thậm chí còn thấp hơn khi so sánh với nhiễu symetric 50% (86.5%). Nguyên nhân của việc chọn ra được nhiều dữ liệu sạch hơn nhưng độ chính xác lại thấp hơn đó chính là do những điểm chọn được không đủ đại diện cho dữ liệu để thực hiện phân tách lớp ở cấp độ “mịn”. Điều này có thể được giải thích là với nhiễu symetric 20% sẽ có 80% dữ liệu sạch trong khi mặc định kích thước coreset của chúng ta chỉ là 50%, mà với phương pháp CRUST được cài đặt bằng thuật toán tham lam thì những điểm có gradient càng gần trung tâm của cụm thì sẽ ưu tiên được chọn hơn, nên các điểm chọn được đa phần sẽ nằm phía bên trong lanh thổ của lớp, chứ không nằm gần biên giới giữa các lớp. Các điểm này sẽ giúp các lớp học cách phân lớp rõ ràng, trong khi các điểm nằm gần biên giới mới giúp mô hình học cách phân lớp mịn. Đối với nhiễu đố xứng 50%, sẽ có 50% dữ liệu sạch, khi đó coreset của chúng ta cũng có kích thước bằng 50% của dữ liệu 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 giới giữa các lớp hơn (ít nhất là sẽ nhiều điểm gần biên hơn nhiễu đố xứng 20%), làm cho mô hình học cách phân lớp mịn tốt hơn dẫn đến hiệu suất cũng cao hơn.

Để chứng minh cho điều này, chúng em đã thực hiện thí nghiệm tăng coreset lên 70% kích thước dữ liệu, sau đó chọn ra 50% điểm (theo kích thước của dữ liệu) có phân bố gradient xa trung tâm nhất của coreset này. Việc này giúp chúng ta chọn ra một subcoreset có kích thước 50% của dữ liệu, nhưng sẽ chọn được nhiều điểm nằm ở phần biên giới giữa các lớp hơn, giúp mô hình học cách phân lớp mịn tốt hơn. So sánh với việc chọn coreset kích thước 50%, cách làm này cải thiện độ chính xác mô hình một cách đáng kể. Cụ thể, độ chính xác cao nhất của cách làm này trên tập validation lên đến 89.97%. Hình 4.5 cho thấy kết quả của cách làm mới tốt hơn.



Hình 4.5: So sánh kết quả huấn luyện nhiễu đối xứng 20% theo phương pháp ban đầu và sau khi cải tiến CIFAR-10

Trên tập CIFAR-10 là vậy, kết quả của nhiễu đối xứng 20% trên tập CIFAR-100 vẫn cao hơn nhiễu đối xứng 50%. Nguyên nhân của kết quả này là do tập CIFAR-100 có đến tận 100 lớp, khó hơn nhiều so với tập CIFAR -10 chỉ có 10 lớp. Đối với bộ dữ liệu này, chỉ việc phân lớp thôi đã là rất khó, độ chính xác với nhiễu đối xứng 20% cũng chỉ đạt 63.9%. 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 mô hình vẫn đang ở giai đoạn phân lớp thôi.

Hướng giải quyết

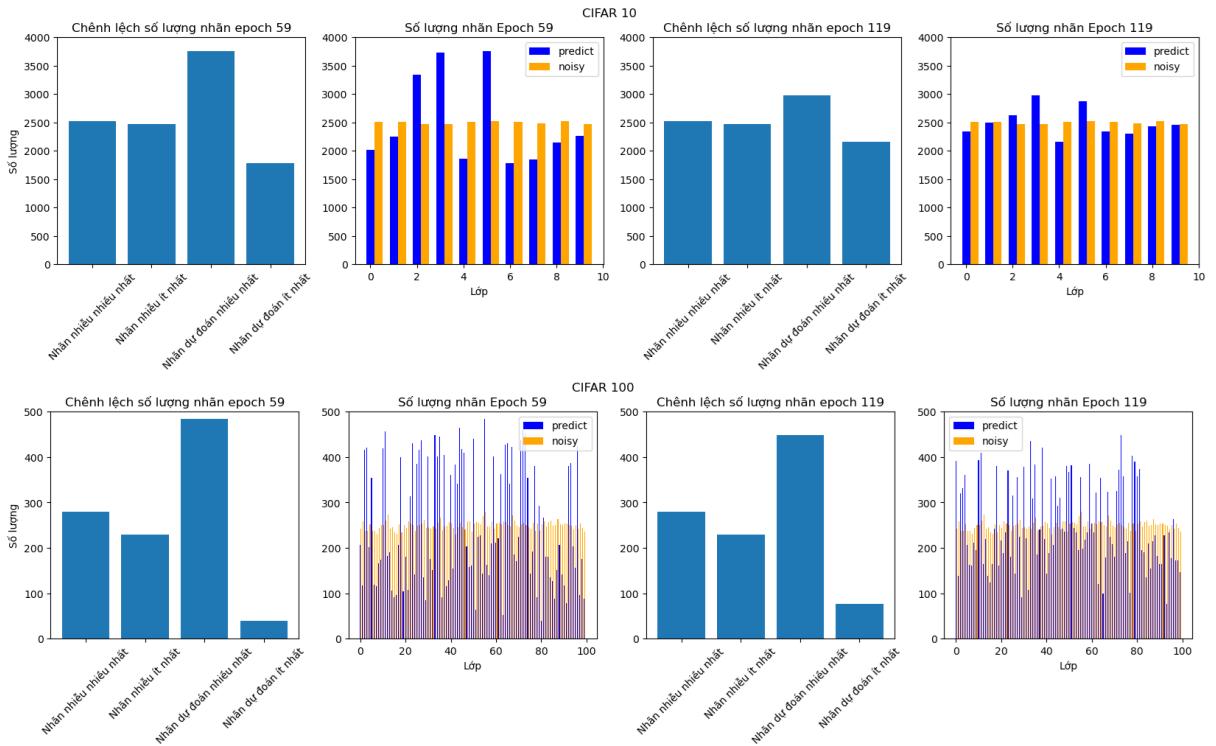
Vấn đề này chỉ xảy ra khi tỷ lệ nhiễu khá thấp, với những trường hợp nhiễu nặng thì việc chọn ra coreset chứa dữ liệu sạch vẫn sẽ mang lại hiệu quả tốt. Trong trường hợp nhiễu thấp, chúng ta có thể tăng kích thước coreset lên để đảm bảo hiệu suất hơn vì khi này coreset sẽ bao hết các điểm dữ liệu sạch và sẽ học được cách phân lớp thô và mịn tốt hơn. Ngoài ra, nếu muốn giữ kích thước coreset nhỏ thì có thể áp dụng cách được trình bày ở mục trước, tăng kích thước coreset nhưng sau đó lại lấy ra những điểm xa trung tâm cụm nhất (Xem kết quả ở hình 4.5)

4.3.3 Phân tích tìm nguyên nhân kết quả của sử dụng nhãn nhiễu cao hơn nhãn dự đoán.

Hình 4.6 cho thấy sự khác nhau giữa số lượng mẫu của các lớp trong coreset khi sử dụng nhãn dự đoán và nhãn nhiễu để gom cụm. Trong khi số lượng mẫu của nhãn nhiễu không chênh lệch nhiều giữa các lớp và không thay đổi qua các epoch, số lượng mẫu của nhãn dự đoán có sự khác biệt rõ ràng. Mặc dù sự chênh lệch về số mẫu này sẽ được cải thiện hơn ở những epoch sau, nhưng khoảng cách giữa lớp có nhiều mẫu nhất và ít mẫu nhất vẫn khá lớn, đặc biệt là tập dữ liệu CIFAR-100. Sự mất cân bằng giữa các lớp khiến cho độ chính xác của mô hình giảm nhiều, thậm chí một số trường hợp sử dụng nhãn dự đoán có Label Accuracy cao hơn những vẫn có độ chính xác thấp hơn.

4.3.4 Phân tích tìm nguyên nhân CRUST có độ chính xác thấp trên tập dữ liệu CIFAR-10 nhiễu sym 80

Hình 4.7 thể hiện sự phân bố gradient của các loại nhiễu khác nhau tại epoch 119. Từ hình này có thể rút ra kết luận rằng với mức độ nhiễu đối xứng 20% và 50% và bất đối xứng 40% trên CIFAR-10 thì gradient phân cụm khá rõ ràng. Đến mức nhiễu đối xứng 80% thì các điểm dữ liệu phân bố cực kỳ hỗn loạn, dữ liệu sạch và nhiễu đan xen nhau. Đây chính là giải thích cho lý do vì sao Label Accuracy của mức độ nhiễu này (xem hình 4.2) chỉ hơn 20% một ít (gần như là bằng tỷ lệ dữ liệu sạch vốn có). Điều

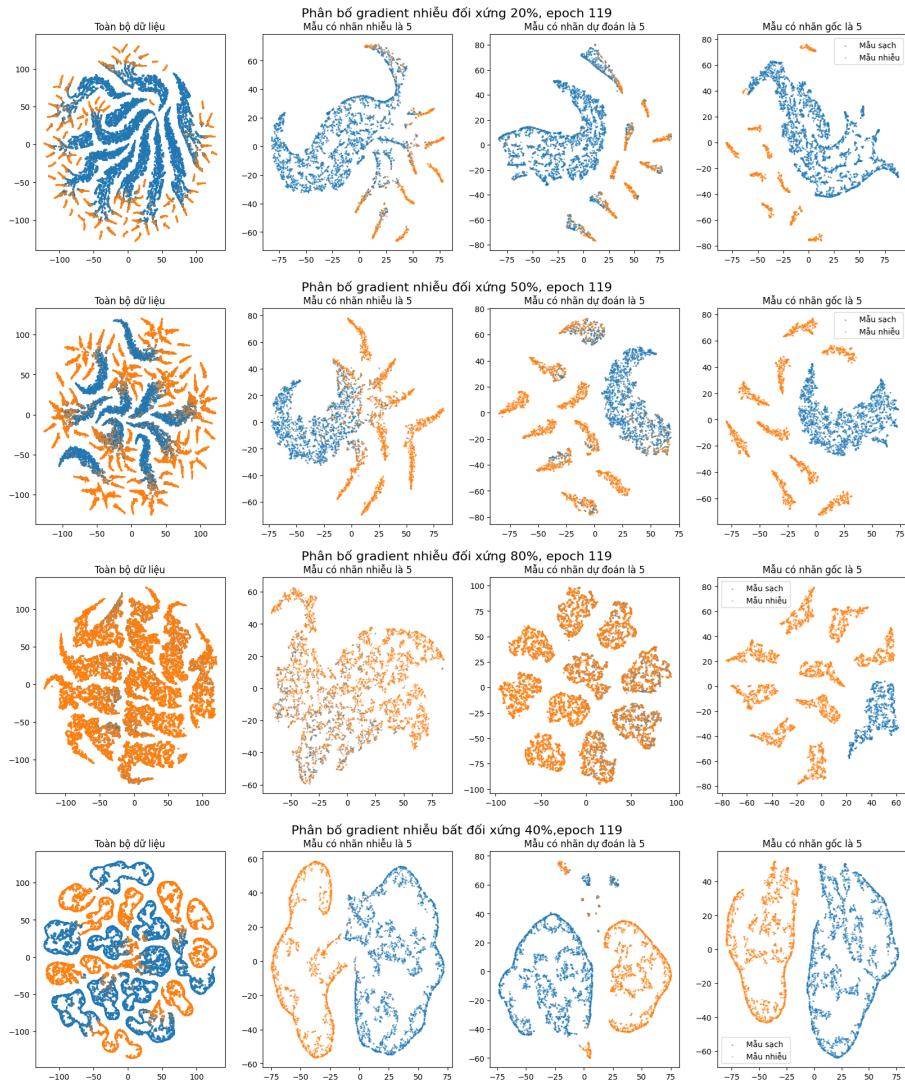


Hình 4.6: So sánh sự không đồng đều giữa số lượng mẫu của lớp trong coreset ở mức nhiễu đối xứng 50%

này cũng trực tiếp dẫn đến việc độ chính xác trên tập validation cực kỳ thấp (36.5%).

4.3.5 Phân tích tìm nguyên nhân CRUST có độ chính xác trên tập validation thấp hơn INCV ở sym 20 trên CIFAR-10 (trên CIFAR-100 thì lại cao hơn)

Nguyên nhân cho kết quả này tương tự với kết quả được trình bày ở mục 4.3.2. Với mức nhiễu sym 20 ở CIFAR-10, coreset chọn được không đủ để đại diện cho dữ liệu để thực hiện phân tách lớp ở cấp độ mịn. Chúng em đã thực hiện phân tích rõ vì sao xảy ra vấn đề này và hướng giải quyết trong mục trên.

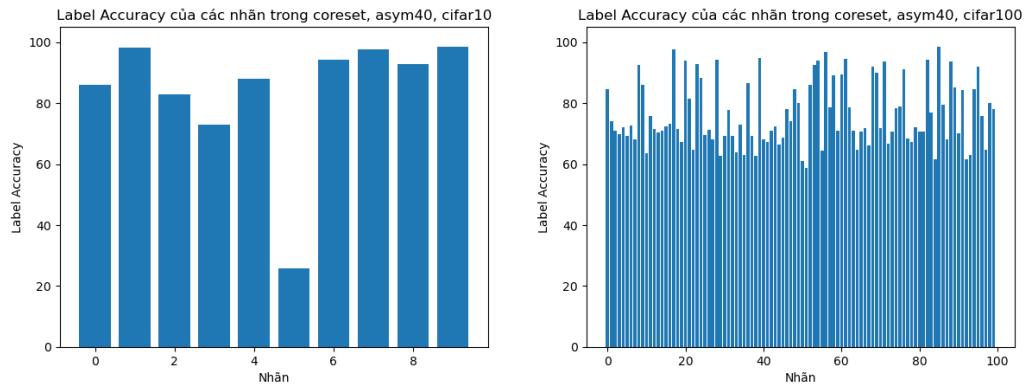


Hình 4.7: Phân bố gradient của các loại nhiễu khác nhau trên CIFAR-10 tại epoch 119

4.3.6 Phân tích tìm nguyên nhân CRUST có độ chính xác trên tập validation thấp hơn INCV ở asym 40 trên CIFAR-10 (trên CIFAR-100 thì lại cao hơn)

Hình 4.8 trực quan Label Accuracy của từng nhãn trong coreset được chọn bởi CRUST với tập dữ liệu CIFAR-10 và CIFAR-100. Có thể thấy rằng với tập dữ liệu CIFAR-10 thì lớp 5 có Label Accuracy cực kỳ thấp, chỉ khoảng 25%, điều này làm mô hình không thể dự đoán tốt trên lớp này, dẫn đến có kết quả kém hơn INCV. Đối với tập dữ liệu CIFAR-100 thì Label Accuracy phân bố đều hơn, có một số lớp cũng cao hơn nhiều so với các lớp khác nhưng không có lớp nào quá thấp. Điều này khiến dự đoán trên CIFAR-100 ổn định hơn, và tốt hơn trên INCV. Nguyên nhân cụ

thể của việc tại sao lại có một số nhãn Label Accuracy cực kỳ thấp chúng em vẫn chưa tìm ra và chúng em cũng chưa có giải pháp nào cho vấn đề này.



Hình 4.8: Label Accuracy của các nhãn trong coresset với mức nhiễu asym 40 trên tập CIFAR-10 và CIFAR-100

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 nghiên cứu phương pháp giải quyết bài toán học có giám sát với dữ liệu nhiễu. Đây là một vấn đề quan trọng và 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. Cụ thể, các điểm dữ liệu có nhãn sạch thường có xu hướng gom cụm lại với nhau, trong khi các điểm dữ liệu có nhãn nhiễu có xu hướng phân tán trong không gian gradient. Giải quyết bài toán này bằng thuật toán k-medoids, tìm ra coreset gồm k điểm là trung tâm của k cụm dữ liệu sạch. Kết quả của cách làm 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. Một số vấn đề chính như sau:

- **Nhận định gradient của dữ liệu nhãn sạch gom cụm và dữ liệu nhãn nhiễu phân không hoàn toàn đúng:** Khi tập dữ liệu có tỉ lệ nhãn nhiễu thấp, nhận định này là đúng. Nhưng khi tỉ lệ nhiễu tăng cao lên, các điểm dữ liệu có nhãn nhiễu cũng có xu hướng gom cụm.
- **Sử dụng nhãn dự đoán để chọn coreset không chắc chắn sẽ mang lại kết quả tốt hơn:** Sự mất cân bằng trong dự đoán các nhãn của mỗi lớp (mô hình tập trung dự đoán vào một số lớp) khiến việc học trên nhãn dự đoán không cho kết quả tốt bằng học trên nhãn nhiễu.
- **Phương pháp không hoạt động tốt khi tỉ lệ nhiễu quá cao:** Khi tỉ lệ nhiễu trở nên quá cao, mô hình rất khó học được thông tin từ dữ liệu, gradient cũng không gom cụm rõ ràng, khiến phương pháp rất khó khăn trong việc chọn được coreset.

Ngoài ra còn một số vấn đề nhỏ khác của phương pháp đã được trình bày trong các phần trước.

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 thuật toán greedy. Thuật toán này giúp tìm coreset dễ dàng hơn, 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 nhiều mà nó cũng có thể mang lại giá trị nhất định nào đó trong các bài toán khác.

Tài liệu tham khảo

References

- [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] Scikit-learn Developers. *Cross-validation: evaluating estimator performance*. Accessed: 2024-06-30. 2024. URL: https://scikit-learn.org/stable/modules/cross_validation.html.
- [4] Bo Han et al. “Co-teaching: Robust training of deep neural networks with extremely noisy labels”. In: *NeurIPS*. 2018, pp. 8535–8545.
- [5] 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.
- [6] Geoffrey Hinton. *Lecture Slides for "Neural Networks for Machine Learning"*. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. 2012.
- [7] Javatpoint. *Gradient Descent in Machine Learning*. Accessed: 2024-06-30. 2023. URL: <https://www.javatpoint.com/gradient-descent-in-machine-learning>.
- [8] 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.

- [9] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA, 2015.
- [10] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).
- [11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).
- [12] Anders Krogh and John A Hertz. “A simple weight decay can improve generalization”. In: *Proc, NeurIPS*. 1992, pp. 950–957.
- [13] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [14] Michel Minoux. “Accelerated greedy algorithms for maximizing submodular set functions”. In: *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg, September 5–9, 1977*. 2005, pp. 234–243.
- [15] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. “coresets for data-efficient training of machine learning models”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6950–6960.
- [16] 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).
- [17] 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.
- [18] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014).

- [19] Mingxing Tan and Quoc Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.
- [20] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).
- [21] Laurence A Wolsey. “An analysis of the greedy algorithm for the submodular set covering problem”. In: *Combinatorica* 2.4 (1982), pp. 385–393.
- [22] Yash. *CNN for MNIST Handwritten Dataset*. Accessed: 2024-06-30. 2024. URL: <https://medium.com/@yash.4198/cnn-for-mnist-handwritten-dataset-cc94d61f6c94>.
- [23] Chiyuan Zhang et al. “Understanding deep learning requires rethinking generalization”. In: *Communications of the ACM* 64 (2016). DOI: 10.1145/3446776.