

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

Nguyễn Ngọc Lan Như - Hoàng Minh Quân

**HUẤN LUYỆN
MẠNG NO-RON NHIỀU TẦNG ẨN
BẰNG THUẬT TOÁN ADAM**

**KHÓA LUẬN TỐT NGHIỆP CỦ NHÂN
CHƯƠNG TRÌNH CHÍNH QUY**

Tp. Hồ Chí Minh, tháng 07/2021

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

Nguyễn Ngọc Lan Như - 1712644

Hoàng Minh Quân - 1712688

**HUẤN LUYỆN
MẠNG NƠ-RON NHIỀU TẦNG ẨN
BẰNG THUẬT TOÁN ADAM**

**KHÓA LUẬN TỐT NGHIỆP CỦ NHÂN
CHƯƠNG TRÌNH CHÍNH QUY**

GIÁO VIÊN HƯỚNG DẪN
ThS. Trần Trung Kiên

Tp. Hồ Chí Minh, tháng 07/2021

Lời cảm ơn

Tôi xin chân thành cảm ơn ...

Mục lục

Lời cảm ơn	i
Mục lục	ii
Danh mục hình ảnh	iv
Danh mục bảng	vii
Tóm tắt	vii
1 Giới thiệu	1
2 Kiến thức nền tảng	9
2.1 Mạng nơ-ron nhiều tầng ẩn	9
2.2 Quá trình huấn luyện mạng nơ-ron nhiều tầng ẩn	14
2.3 “Gradient Descent”	17
2.3.1 “Batch” Gradient Descent	19
2.3.2 “Minibatch” Gradient Descent	20
2.4 “Lan truyền ngược” (Backpropagation)	22
3 Huấn luyện mạng nơ-ron nhiều tầng ẩn bằng thuật toán Adam	26
3.1 Thuật toán Gradient Descent với Momentum	27
3.2 Thuật toán Gradient Descent với tỉ lệ học thích ứng	31

3.3	Thuật toán Adam	38
4	Các kết quả thí nghiệm	42
4.1	Các thiết lập thí nghiệm	42
4.2	Các kết quả thí nghiệm	45
4.2.1	Kết quả của thuật toán cài đặt so với bài báo . . .	45
4.2.2	Phân tích độ lớn bước cập nhật của các trọng số trong rãnh hẹp	51
4.2.3	Phân tích đường đi trong rãnh hẹp có hướng trùng và không trùng với trọng số	53
4.2.4	Vấn đề đặc trưng thừa và nhiễu	54
4.2.5	Huấn luyện mô hình VGG16	57
4.2.6	Huấn luyện mô hình ngôn ngữ	59
5	Kết luận và hướng phát triển	61
5.1	Kết luận	61
5.2	Hướng phát triển	61
Tài liệu tham khảo		62
A	Các siêu tham số của các thí nghiệm	66

Danh mục hình ảnh

1.1	Minh họa quá trình mạng nơ-ron “học” từ các đặc trưng đơn giản, như các điểm ảnh, tạo thành các đặc trưng phức tạp hơn.[3]	2
1.2	Minh họa cho mô hình overfit và underfit. (Nguồn: https://www.analyticsvidhya.com/blog/2020/02/underfitting-overfitting-best-fitting-machine-learning)	3
1.3	Đồ thị contour cho những điểm critical point: vùng bằng phẳng có một cực tiểu (trái), điểm yên ngựa (giữa), vùng rãnh hẹp (phải).	4
1.4	Bề mặt lỗi của mô hình ResNet-110 (trái) và bề mặt lỗi của mô hình ResNet-56 (phải).[17]	5
1.5	Minh họa cho hướng đi của gradient trong trường hợp di chuyển trong rãnh hẹp. (Nguồn: https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/)	5
2.1	Mô hình mạng nơ-ron nhân tạo với tầng nhập (màu cam), tầng ẩn (màu xanh dương) và tầng xuất (màu xanh lá).	10
2.2	Minh họa cách hoạt động của một nơ-ron. (Nguồn: https://wiki.pathmind.com/neural-network)	11
2.3	Mô hình mạng nơ-ron nhiều tầng ẩn với tầng nhập (màu cam), các tầng ẩn (màu xanh dương) và tầng xuất (màu xanh lá)	12
2.4	Quá trình mạng nơ-ron trích xuất các đặc trưng của dữ liệu.[12]	13

2.5	Minh họa điểm cực tiểu (a), cực đại (b) và điểm yên ngựa (c) trong không gian ba chiều.	15
2.6	Minh họa quá trình tìm đến điểm cực tiểu của thuật toán Gradient Descent với đường màu xanh dương biểu diễn giá trị của hàm chi phí, đường đứt đoạn màu đỏ thể hiện phương và độ lớn của gradient.	17
2.7	So sánh đường đi của Gradient Descent (trái) và Stochastic Gradient Descent (phải).	21
2.8	Minh họa mô hình mạng nơ-ron đơn giản gồm ba tầng: 1 tầng nhập (màu cam), 2 tầng ẩn (màu lam) và 1 tầng xuất (màu lục). Trong đó, w_i là các trọng số liên kết, b_i là hệ số bias tương ứng của từng nơ-ron và a_i là giá trị kích hoạt của nơ-ron.	24
3.1	Minh họa cách Momentum triệt tiêu hướng có độ dốc lớn. (Nguồn: https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/)	30
3.2	Xấp xỉ bậc hai (đường đứt khúc màu xanh nhạt) của hàm lỗi (đường màu xanh đậm) và bước cập nhật tiếp theo của phương pháp Newton.	32
3.3	Cực tiểu của xấp xỉ bậc hai cung cấp một bước cập nhật không tối ưu.	35
4.1	Kết quả thí nghiệm Multi-layer Neural Network giữa thuật toán mà chúng tôi cài đặt so với bài báo. Bên trái: kết quả mà bài báo công bố. Bên phải: kết quả do chúng tôi tự cài đặt lại.	46
4.2	Kết quả thí nghiệm Convolutional Neural Network giữa thuật toán mà chúng tôi cài đặt so với bài báo. (a): kết quả mà bài báo công bố. (b): kết quả do chúng tôi tự cài đặt lại. .	48

4.3	Kết quả thí nghiệm Convolutional Neural Network giữa thuật toán mà chúng tôi cài đặt với bộ siêu tham số mô phỏng kết quả của bài báo.	49
4.4	Kết quả thí nghiệm Convolutional Neural Network mà không thực hiện làm trắng ảnh.	50
4.5	Đường đi, độ lớn bước cập nhật theo từng chiều, và độ lỗi của các thuật toán trong quá trình tối ưu hàm giả lập. . .	52
4.6	Đường đi của các thuật toán trong rãnh hẹp có hướng trùng với trọng số (a) và không trùng với trọng số (b).	54
4.7	Đường đi của các thuật toán trong bề mặt lỗi (trái) và độ lỗi (phải) của từng thuật toán trong trường hợp đặc trưng thừa.	55
4.8	Đường đi của các thuật toán trong bề mặt lỗi (trái) và độ lỗi (phải) của từng thuật toán trong trường hợp đặc trưng thừa.	57
4.9	Đường đi của các thuật toán trong bề mặt lỗi (trái) và độ lỗi (phải) của từng thuật toán trong trường hợp đặc trưng thừa.	58
4.10	Mô tả kết quả thí nghiệm của mô hình ngôn ngữ trên tập dữ liệu Penn Treebank gồm kết quả trên tập huấn luyện (trái) và tập kiểm thử (phải).	59

Danh mục bảng

4.1	Khoảng giá trị để dò tìm siêu tham số tốt nhất của các thuật toán tối ưu.	44
4.2	Kết quả và thời gian thực hiện một epoch của các thuật toán.	50
A.1	Các siêu tham số được sử dụng trong thí nghiệm Multi-layer Neural Network.	66
A.2	Các siêu tham số được sử dụng trong thí nghiệm Convolutional Neural Network.	67
A.3	Các siêu tham số được sử dụng trong thí nghiệm mô hình ngôn ngữ.	67

Tóm tắt

Trong nhiều năm trở lại đây, các mô hình mạng nơ-ron nhiều tầng ẩn đã tạo nên những bước cải tiến lớn trong vô số lĩnh vực khoa học khác nhau. Để đạt được những kết quả đó, tất cả mô hình đều yêu cầu một quá trình điều chỉnh bộ trọng số để mô hình có thể thích ứng với dữ liệu huấn luyện và dự đoán trên dữ liệu mới một cách chính xác. Vì vậy, một phương pháp đi tìm bộ trọng số hiệu quả sẽ cải thiện độ chính xác của mô hình, rút ngắn thời gian huấn luyện, từ đó nâng cao tính ứng dụng của các mô hình mạng nơ-ron nhiều tầng ẩn.

Khóa luận tập trung tìm hiểu về những khó khăn trong việc huấn luyện mạng nơ-ron nhiều tầng ẩn và cách mà các thuật toán tối ưu giải quyết những khó khăn đó, với trọng tâm là thuật toán Adam. Thuật toán Adam được giới thiệu trong bài báo "Adam: A Method for Stochastic Optimization" công bố tại hội nghị "International Conference on Learning Representation 2015". Ý tưởng của thuật toán là kết hợp hai hướng tiếp cận: (1) sử dụng quán tính để tăng tốc và giảm dao động, và (2) thích ứng lượng cập nhật cho từng trọng số; với mục tiêu kế thừa được những điểm mạnh và đồng thời cải thiện những điểm yếu của mỗi hướng tiếp cận.

Kết quả đạt được của khóa luận là cài đặt lại thuật toán Adam cùng các thuật toán liên quan và tái tạo được một phần kết quả bài báo gốc. Khóa luận cũng thực hiện thêm các thí nghiệm nhằm phân tích và làm rõ cách mỗi phương pháp giải quyết các khó khăn trong việc huấn luyện mạng nơ-ron nhiều tầng ẩn.

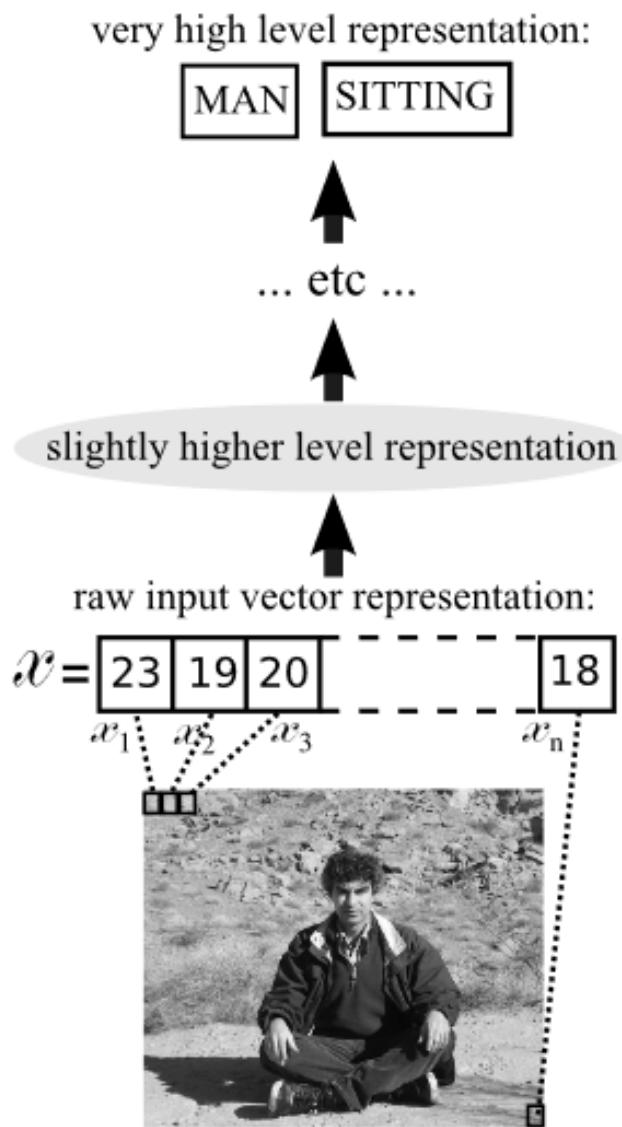
Chương 1

Giới thiệu

Việc sử dụng mạng nơ-ron nhiều tầng ẩn trong các ứng dụng trí tuệ nhân tạo ngày càng chiếm một vị trí quan trọng khi mà các bài toán khó của học máy truyền thống trong xử lý ảnh và video, xử lý ngôn ngữ tự nhiên như: dịch máy tự động, nhận diện mặt người, phát hiện đồ vật,... đã phần nào được giải quyết và ngày càng được cải tiến.

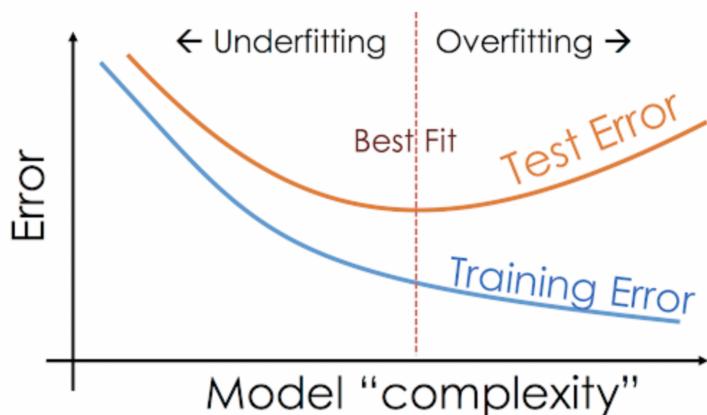
Mạng nơ-ron nhiều tầng ẩn là một mạng nơ-ron nhân tạo gồm ba loại tầng chính: tầng nhập, các tầng ẩn, và tầng xuất. Mỗi tầng được cấu thành từ nhiều đơn vị tính toán mà ta gọi là các nơ-ron. Các nơ-ron được liên kết với nơ-ron ở tầng tiếp theo bằng các liên kết có trọng số đại diện cho mức độ quan trọng của tín hiệu từ nơ-ron này sang nơ-ron khác. Số lượng trọng số sẽ tăng theo số lượng tầng ẩn có trong mạng. Việc có nhiều tầng ẩn cũng là một trong những lý do giúp mạng nơ-ron nhiều tầng ẩn có thể giải quyết các bài toán khó mà các thuật toán học máy truyền thống không giải quyết được. Mặc dù vẫn còn nhiều ý kiến về việc sử dụng nhiều tầng ẩn có thực sự cần thiết như công trình của Jimmy Lei Ba[1] chứng minh được rằng một mạng nơ-ron nông (shallow network) vẫn có thể xấp xỉ kết quả của một mạng nơ-ron nhiều tầng ẩn. Tuy nhiên, để đạt được kết quả đó, bài báo phải sử dụng một tập dữ liệu có kích thước rất lớn và một mạng nơ-ron nhiều tầng ẩn có độ chính xác khá cao. Ngoài việc mạng nơ-ron nhiều tầng ẩn có thể giúp cho quá trình thiết kế mô hình cho

từng bài toán dễ dàng hơn[20] thì sức mạnh rút trích đặc trưng tự động cũng đã được chỉ ra trong nhiều bài báo khác: Yoshua Bengio và Yann Lecun lý giải rằng với nhiều tầng ẩn, mạng nơ-ron có thể “học” được từ đặc trưng đơn giản (low-level features) và tạo thành các đặc trưng phức tạp hơn (higher-level features)[4] (hình 1.1); một công trình khác của Yoshua Bengio cho thấy sức mạnh của mạng nơ-ron nhiều tầng ẩn khi nó có thể giải quyết các bài toán mà các mạng nơ-ron ít tầng ẩn hơn không thể [3].



Hình 1.1: Minh họa quá trình mạng nơ-ron “học” từ các đặc trưng đơn giản, như các điểm ảnh, tạo thành các đặc trưng phức tạp hơn.[3]

Tuy nhiên, mặc dù mạng nơ-ron có càng nhiều tầng ẩn sẽ khiến cho độ chính xác trong tập huấn luyện ngày càng tăng nhưng độ chính xác trong tập kiểm thử ngày càng giảm. Đây là biểu hiện của việc mô hình bị overfit, đồng nghĩa với việc mô hình chỉ “ghi nhớ” tập huấn luyện nhưng không thể tổng quát hóa những đặc trưng đã học để tiến hành đưa ra dự đoán trên tập kiểm thử. Ngược lại, nếu mạng nơ-ron quá đơn giản, có thể có ít tầng ẩn hoặc mỗi tầng ẩn có số lượng nơ-ron ít, thì mô hình đó không đủ khả năng để rút trích ra những đặc trưng có ý nghĩa mà mô hình có thể học khiến cho độ chính xác ở cả hai tập dữ liệu huấn luyện và kiểm thử đều giảm (hình 1.2).

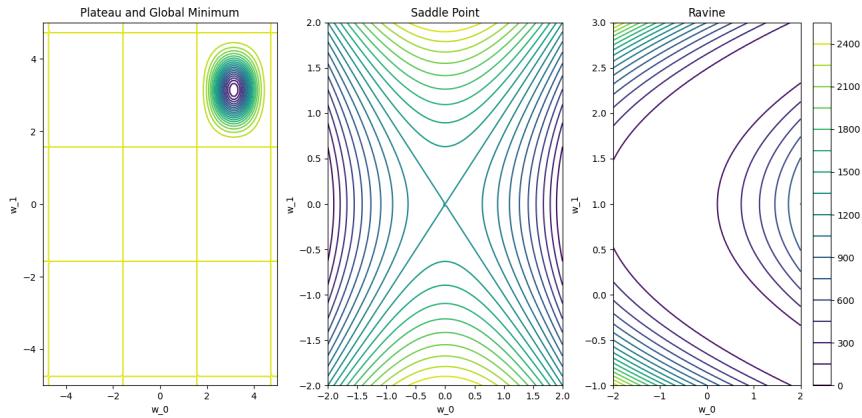


Hình 1.2: Minh họa cho mô hình overfit và underfit. (Nguồn: <https://www.analyticsvidhya.com/blog/2020/02/underfitting-overfitting-best-fitting-machine-learning>)

Một mạng nơ-ron nhiều tầng ẩn được xem là có khả năng tổng quát hoá tốt trên một bài toán khi sự sai khác giữa giá trị dự đoán của mạng nơ-ron và giá trị nhãn của dữ liệu ngoài tập huấn luyện là đủ nhỏ (trong bài toán huấn luyện có giám sát). Để đạt được kết quả đó, mạng nơ-ron nhiều tầng ẩn cần đi tìm một bộ trọng số phù hợp cho từng bài toán cụ thể. Việc đi tìm bộ trọng số này được thực hiện thông qua quá trình tối ưu hoá mạng nơ-ron nhiều tầng ẩn.

Bài toán tối ưu hoá mạng nơ-ron nhiều tầng ẩn nhận dữ liệu nhập là hàm chi phí nhận bộ trọng số của mạng nơ-ron nhiều tầng ẩn làm tham số.

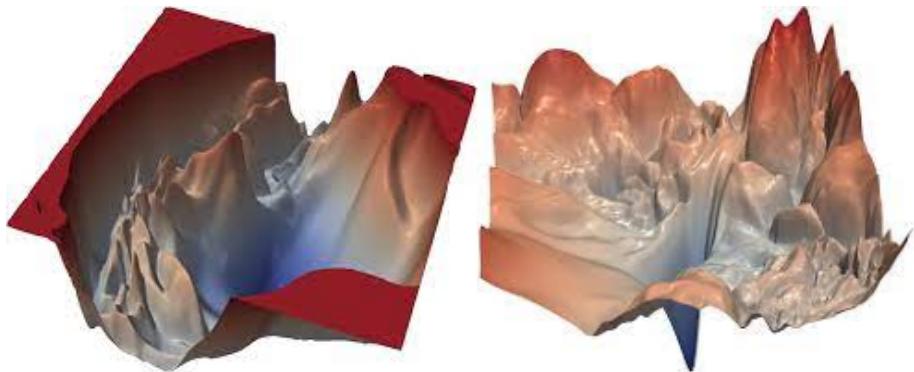
Hàm chi phí cho biết sự sai lệch giữa kết quả dự đoán của mạng nơ-ron so với giá trị đúng. Giá trị sai lệch này còn được gọi là độ lỗi. Sau quá trình tối ưu ta mong muốn có được một bộ trọng số của mạng nơ-ron nhiều tầng ẩn cho độ lỗi trong cả hai tập dữ liệu huấn luyện và tập dữ liệu kiểm tra là đủ nhỏ.



Hình 1.3: Đồ thị contour cho những điểm critical point: vùng băng phẳng có một cực tiểu (trái), điểm yên ngựa (giữa), và vùng rãnh hẹp (phải).

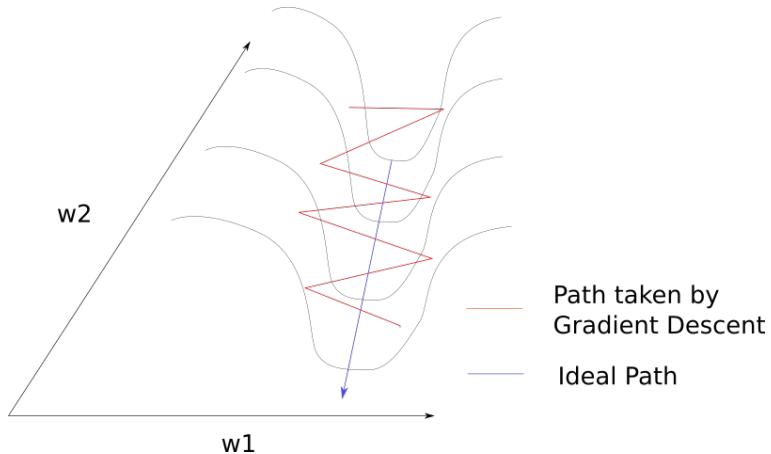
Việc tối ưu hóa mạng nơ-ron có thể hiểu là quá trình đi tìm cực tiểu của hàm chi phí bằng cách thay đổi giá trị của bộ trọng số. Từ đó, ta cũng có thể hiểu quá trình tối ưu hóa mạng nơ-ron là quá trình di chuyển trong bề mặt lỗi dựa (Hình 1.4) trên hướng của véc-tơ đạo hàm riêng, còn được gọi là gradient. Tuy nhiên, việc di chuyển trong bề mặt lỗi gặp nhiều khó khăn. Li Hao và cộng sự[17] cho thấy rằng sự phức tạp của bề mặt lỗi ngày càng tăng khi số tầng ẩn trong mạng nơ-ron ngày càng tăng. Sự hỗn loạn này được cấu thành từ nhiều “critical point” (“critical point” là những điểm có gradient bằng 0 như cực tiểu địa phương, điểm yên ngựa) hợp với các vùng băng phẳng, và vùng rãnh hẹp (Hình 1.3).

Các điểm cực tiểu địa phương từng được xem như là nguyên nhân chính gây ra sự khó khăn trong việc tối ưu hóa mạng nơ-ron có nhiều tầng ẩn, nhưng nghiên cứu của Quynh Nguyen và Matthias Hein[19] đã chỉ ra rằng việc đi tìm cực tiểu toàn cục thật sự có thể mất nhiều thời gian nhưng cho



Hình 1.4: Bề mặt lỗi của mô hình ResNet-110 (trái) và bề mặt lỗi của mô hình ResNet-56 (phải).[17]

hiệu quả không đáng kể. Yann N. Dauphin cũng cho rằng khi số lượng tầng ẩn quá lớn thì phần lớn các điểm critical point sẽ là điểm yên ngựa, và các điểm cực tiểu tìm được đều có độ lỗi ngang với cực tiểu toàn cục. Hơn nữa, các điểm yên ngựa thường được bao bọc bởi một vùng gần như bằng phẳng, là vùng mà tại đó đạo hàm xấp xỉ gần bằng 0, làm chậm tốc độ của đa số các thuật toán tối ưu thường được sử dụng hiện nay.



Hình 1.5: Minh họa cho hướng đi của gradient trong trường hợp di chuyển trong rãnh hẹp. (Nguồn: <https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/>)

Ngoài các điểm yên ngựa, bề mặt lỗi còn đặc trưng bởi độ cong tại mỗi hướng, tương ứng với một trọng số của mạng nơ-ron nhiều tầng ẩn (hình 1.5). Một hướng có độ cong thấp (low-curvature, như hướng w_2 trong

hình) đồng nghĩa với gradient sẽ ít thay đổi và ngược lại, một hướng có độ cong cao (high-curvature, như hướng w_1 trong hình) sẽ có gradient thay đổi nhiều. Vùng được đồng thời tạo bởi các hướng có độ cong cao và độ cong thấp được gọi là vùng rãnh hẹp. Các vùng này được tạo ra do ảnh hưởng của các trọng số w_1 và w_2 lên độ lỗi là không giống nhau. Một w_1 có giá trị gấp 10 lần giá trị của w_2 sẽ cho tín hiệu của các nơ-ron được liên kết bằng w_1 ảnh hưởng lên độ lỗi gấp 10 lần tín hiệu của các nơ-ron được liên kết bằng w_2 , dẫn đến độ cong theo hướng w_1 sẽ cao hơn độ cong theo hướng w_2 gấp 10 lần. Sự cách biệt lớn trong độ lớn của các giá trị trọng số này tạo ra vùng rãnh hẹp. Hiện tượng này thường xuyên xảy ra trong mạng nơ-ron do tín hiệu của các nơ-ron khác nhau có mức độ ảnh hưởng khác nhau đến quá trình dự đoán, và càng nghiêm trọng hơn trong các mạng nơ-ron nhiều tầng ẩn.

Phương pháp đầu tiên được dùng trong huấn luyện mạng nơ-ron thực hiện bước từng bước nhỏ theo hướng của véc-tơ đạo hàm riêng gradient tìm đến điểm cho độ lỗi nhỏ hơn được gọi là “Gradient Descent” (GD). Tuy nhiên, phương pháp này tốn nhiều thời gian trong việc tính toán véc-tơ gradient tại điểm hiện tại. Các đạo hàm riêng theo từng trọng số trong gradient có được từ việc lấy đạo hàm của trung bình độ lỗi trên toàn bộ điểm dữ liệu. Với kích thước tập dữ liệu cần cho huấn luyện mạng nơ-ron nhiều tầng ẩn càng lớn thì việc tính toán gradient tại mỗi bước lại càng mất nhiều thời gian. Để cải thiện vấn đề này, một phương pháp khác không sử dụng toàn bộ điểm dữ liệu để thực hiện một bước cập nhật. Thay vào đó, chỉ một tập con lấy ngẫu nhiên được sử dụng để tính véc-tơ gradient tại mỗi bước. Phương pháp này được gọi là “Stochastic Gradient Descent” (SGD). Vì thực hiện tính toán trên một tập con dữ liệu nên hướng gradient của SGD chỉ là một xấp xỉ của hướng gradient thật của bề mặt lỗi có độ sai số tỉ lệ nghịch với kích thước tập con.

Tuy nhiên, SGD thực hiện một bước nhảy tỉ lệ với độ lớn của gradient nên gặp khó khăn khi di chuyển trong các vùng rãnh hẹp và vùng có độ nghiêng nhỏ. Trong những vùng này, ta mong muốn độ lớn bước cập nhật

tỉ lệ nghịch với độ lớn của gradient. Vì vùng có độ lớn gradient nhỏ thì một bước đi dài sẽ giúp giảm thời gian di chuyển trong khi những vùng có gradient lớn cần bước nhữngh bước nhỏ hơn. Sự điều chỉnh này trong bước cập nhật của SGD cho ta thuật toán “SGD với Momentum”, hay “Momentum”. Thuật toán Momentum cập nhật trọng số dựa trên gradient của bước hiện tại và gradient của những bước cập nhật gần nhất. Từ đó, một lượng “quán tính” được thêm vào giúp triệt tiêu các hướng có độ cong cao và cho phép một bước cập nhật lớn hơn tại hướng có độ dốc thấp. Bước cải tiến này không chỉ giúp thuật toán Momentum thực hiện những bước cập nhật có ý nghĩa trong vùng rãnh hẹp và vùng có độ nghiêng nhỏ mà còn giúp xấp xỉ gradient trong SGD gần hơn với gradient thật của bề mặt lỗi từ đó tăng độ chính xác trong hướng di chuyển của thuật toán.

Thuật toán SGD với Momentum giúp cải thiện tốc độ huấn luyện mạng nơ-ron đáng kể và vẫn được sử dụng trong một số mạng nơ-ron nhiều tầng ẩn hiện nay. Mặc dù vậy thuật toán vẫn chưa giải quyết được các khó khăn vì lượng "quán tính" được thêm vào không có nhiều hiệu quả trong những vùng rãnh hẹp mà độ lớn của đạo hàm riêng tại mỗi hướng cập nhật rất khác nhau. Richard S. Sutton cho rằng để có được một bước cập nhật có hiệu quả thì tỉ lệ học cần phải được thay đổi ứng với độ cong của từng hướng [28]. Đây là một trong những lý do các phương pháp bậc hai được sử dụng. Các phương pháp bậc hai là những phương pháp sử dụng ma trận Hessian — là ma trận đạo hàm riêng bậc hai theo từng cặp hướng — để ước lượng độ cong tại một điểm trên bề mặt lỗi. Từ các ước lượng bậc hai, các phương pháp bậc hai cho bước cập nhật tốt hơn các phương pháp chỉ sử dụng gradient.

Lấy ý tưởng từ những phương pháp trên, Diederik P. Kingma và Jimmy Lei Ba đã đề xuất một thuật toán tối ưu bậc nhất cho mạng nơ-ron nhiều tầng ẩn với tốc độ cao hơn so với hầu hết những thuật toán trước đó[14]. Công trình này, “Adam, A Method for Stochastic Optimization”, được công bố tại hội nghị “International Conference on Learning Representation 2015”. Ý tưởng của thuật toán là kết hợp hai hướng tiếp cận trước đó: (1)

sử dụng quán tính để tăng tốc và giảm dao động, và (2) thích ứng tỉ lệ học cho từng trọng số. Thuật toán Adam sử dụng trung bình chạy để tích tụ quán tính kết hợp với phương pháp xấp xỉ ma trận Hessian của phương pháp tỉ lệ học thích ứng. Sự kết hợp này cho phép thực hiện các bước cập nhật tốt hơn tại vùng có độ lớn đạo hàm riêng rất khác nhau. Đồng thời cũng cải thiện độ lớn của bước cập nhật khi véc-tơ gradient không song song với trục trọng số. Ngoài ra, Adam còn thực hiện “bias-correction” giúp thuật toán không bị “bias” về giá trị khởi tạo cho phép thực hiện các bước cập nhật tốt hơn ngay những bước đầu tiên. Trong khóa luận này, chúng tôi tìm hiểu và cài đặt lại thuật toán Adam cùng với các thuật toán liên quan. Chúng tôi cũng thực hiện thêm các thí nghiệm nhằm phân tích và làm rõ cách mỗi phương pháp giải quyết các khó khăn trong việc huấn luyện mạng nơ-ron nhiều tầng ẩn. Ngoài ra, chúng tôi cũng sử dụng tính toán song song trên GPU để tăng tốc độ xử lí cho các thí nghiệm.

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

- Chương 2 giới thiệu sơ lược về mạng nơ-ron nhiều tầng ẩn và quá trình huấn luyện, cũng như nguyên lý của thuật toán Gradient Descent.
- Chương 3 trình bày về thuật toán Adam và các thuật toán nền tảng. Chương này là phần chính của khóa luận.
- Chương 4 trình bày về các thí nghiệm về nguyên lý cũng như thực tiễn để phân tích các tính chất của thuật toán Adam.
- Cuối cùng, chương 5 trình bày kết luận và hướng phát triển.

Chương 2

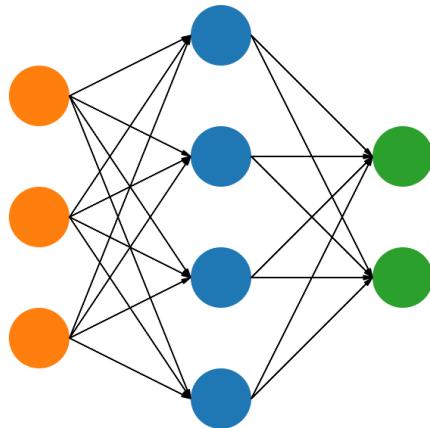
Kiến thức nền tảng

Trong chương này, chúng tôi trình bày các kiến thức nền tảng được sử dụng trong khóa luận. Đầu tiên, chúng tôi giới thiệu tổng quan về mạng nơ-ron nhiều tầng ẩn cũng như quá trình huấn luyện một mô hình mạng nơ-ron và những khó khăn cần khắc phục. Tiếp theo chúng tôi giới thiệu thuật toán tối ưu Gradient Descent và hai phiên bản của thuật toán đó là “Batch Gradient Descent” (BGD) và “Minibatch Gradient Descent” (MGD) là nền tảng cho các thuật toán tối ưu sau này nhằm giải quyết các khó khăn trong huấn luyện mạng nơ-ron.

2.1 Mạng nơ-ron nhiều tầng ẩn

Ngay từ những ngày đầu tiên của điện toán, những nhà khoa học đã hướng đến việc tạo ra những chiếc máy tính có thể thực hiện các tác vụ đòi hỏi khả năng suy nghĩ như bộ não con người. Mục tiêu cuối cùng của những chiếc máy tính này là thay thế con người thực hiện các tác vụ phức tạp như nhận diện vật thể trong hình ảnh, hiểu được ý nghĩa của ngôn ngữ tự nhiên, cũng như chơi được những trò chơi trí tuệ như cờ vua và cờ vây. Hướng nghiên cứu những phương pháp cung cấp cho máy tính khả năng suy nghĩ như con người được gọi là “trí tuệ nhân tạo” (artificial intelligence).

Các phương pháp trí tuệ nhân tạo truyền thống có thể giải quyết được một số bài toán như tìm đường đi tối ưu một cách nhanh hơn, giải một số trò chơi như 8-puzzle, và thậm chí có thể đánh cờ vua ở trình độ Grandmaster (Đại kiện tướng) [7]. Tuy nhiên, những phương pháp này chưa đủ sức tự giải quyết được các bài toán cấp cao hơn như nhận diện thông tin trong ảnh và văn bản. Lí do là vì trí tuệ nhân tạo truyền thống cần con người tạo ra các đặc trưng thủ công (hand-crafted features) dựa trên những hiểu biết của con người về dữ liệu và bài toán đó. Một hướng nghiên cứu hướng tới việc máy tính có thể tự động rút trích những đặc trưng đó một cách tự động thông qua việc huấn luyện một mô hình thích ứng với dữ liệu được gọi là học máy (machine learning). Các mô hình học máy cho phép máy tính có thể chuyển đổi các đặc trưng đơn giản ban đầu thành “kiến thức” để hỗ trợ trong việc dự đoán. Nhờ có những “kiến thức” này mà các mô hình máy học có thể dùng để ứng dụng trong nhiều lĩnh vực khác nhau.

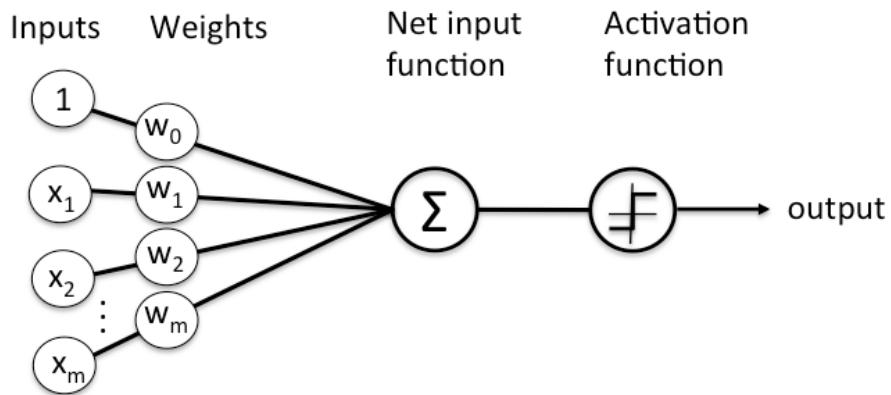


Hình 2.1: Mô hình mạng nơ-ron nhân tạo với tầng nhập (màu cam), tầng ẩn (màu xanh dương) và tầng xuất (màu xanh lá).

Trong số những phương pháp học máy được phát triển, một nhóm phương pháp tên là mạng nơ-ron nhân tạo (artificial neural network) (Hình 2.1) cố gắng mô phỏng lại cách các nơ-ron thần kinh trong não người liên kết với nhau để xử lý tín hiệu đầu vào từ các giác quan và truyền tín hiệu

đã qua xử lý cho các nơ-ron tiếp theo. Sự đòi hỏi lớn về đơn vị tính toán là khó khăn khiến cho các thuật toán học máy truyền thống không thể giải quyết tốt các bài toán mà hàm số biểu diễn chúng phụ thuộc vào nhiều tham số. Thêm nữa, các thuật toán học máy truyền thống không hoạt động tốt đối với những dữ liệu mới, nên việc ứng dụng các mô hình này vào thực tế bị hạn chế. Mạng nơ-ron cho phép kết hợp nhiều hàm phức tạp bằng các nơ-ron được liên kết với nhau. Sự kết hợp này cho phép thực hiện những dự đoán vừa phụ thuộc vào dữ liệu huấn luyện vừa có thể tự tổng quát hoá cho dữ liệu mới.

Trong môi trường máy tính, mỗi “nơ-ron” nhân tạo là một hàm số ánh xạ các tín hiệu đầu vào tới một tín hiệu đầu ra (Hình 2.2). Mỗi tín hiệu đầu vào sẽ được nhân với trọng số tương ứng. Tổng của các tín hiệu này sẽ được cộng với một hệ số bias riêng biệt của tầng nơ-ron đó trước khi đi qua một “hàm kích hoạt” (activation function). Hàm kích hoạt đóng vai trò quyết định việc tín hiệu sẽ được biến đổi như thế nào sau khi đi qua nơ-ron đó.

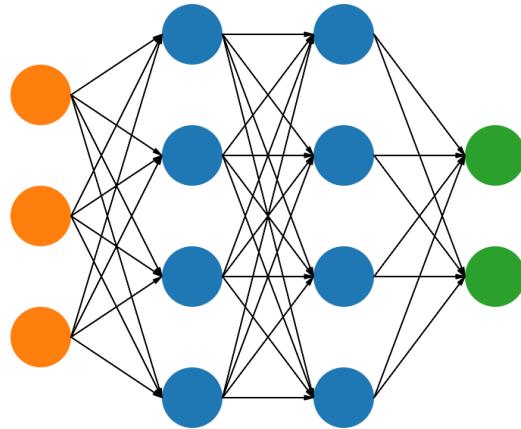


Hình 2.2: Minh họa cách hoạt động của một nơ-ron. (Nguồn: <https://wiki.pathmind.com/neural-network>)

Một hàm số càng có nhiều biến số sẽ càng cần nhiều nơ-ron để biểu diễn lại hàm số đó, đồng nghĩa với việc cần một lượng dữ liệu rất lớn để “học” được hàm số đó. Sử dụng mạng nơ-ron nhiều tầng ẩn giúp biểu diễn một hàm số có độ phức tạp tương đương nhưng với số lượng nơ-ron ít hơn

rất nhiều, theo đó là số lượng dữ liệu cần để huấn luyện cũng giảm đi đáng kể[4].

“Mạng nơ-ron nhiều tầng ẩn” (deep neural network) mở rộng mô hình theo chiều sâu, tăng số lượng tầng ẩn trung gian thay vì tăng số lượng nơ-ron cho một tầng ẩn duy nhất. Hình 2.3 mô tả một mạng nơ-ron nhiều tầng ẩn đơn giản với một tầng nhập, hai tầng ẩn và một tầng xuất.



Hình 2.3: Mô hình mạng nơ-ron nhiều tầng ẩn với tầng nhập (màu cam), các tầng ẩn (màu xanh dương) và tầng xuất (màu xanh lá)

Quá trình truyền thẳng dữ liệu qua mạng nơ-ron nhiều tầng ẩn cũng có thể được biểu diễn dưới dạng các hàm số được xâu chuỗi với nhau. Xét mạng nơ-ron có 2 tầng ẩn ở Hình 2.3, chúng ta có thể biểu diễn mô hình đó bằng công thức $\hat{y} = f^{(2)}(f^{(1)}(x))$ với $f^{(1)}$ là tầng ẩn thứ nhất và $f^{(2)}$ là tầng ẩn thứ 2, và \hat{y} sẽ là giá trị cuối cùng mà mô hình dự đoán được từ dữ liệu đầu vào x . Qua mỗi hàm $f^{(i)}$ trung gian, dữ liệu của tầng trước sẽ bị biến đổi phi tuyến, hay ánh xạ dữ liệu sang chiều không gian khác, và truyền đến tầng tiếp theo.



Hình 2.4: Quá trình mạng nơ-ron trích xuất các đặc trưng của dữ liệu.[12]

Ngoài trích xuất được thêm thông tin, các tầng ẩn còn có khả năng kết hợp các đặc trưng đơn giản ở tầng trước đó thành các đặc trưng cấp cao hơn. Hình 2.4 cho thấy ở tầng ẩn đầu tiên, mô hình kết hợp các giá trị điểm ảnh thành các đường nét đơn giản. Sau đó ở tầng tiếp theo, các đường nét được kết hợp thành các hình dạng. Từ các hình dạng, mạng nơ-ron hình thành những vật thể hoàn chỉnh hơn và đưa ra dự đoán. Như vậy, chúng ta có thể thấy được rằng, với càng nhiều tầng ẩn, mô hình mạng nơ-ron sẽ càng rút trích thông tin tốt hơn, từ đó tăng cường khả năng xử lý các bài toán phức tạp.

2.2 Quá trình huấn luyện mạng nơ-ron nhiều tầng ẩn

Xét một bài toán học có giám sát (supervised learning), chúng ta có tập dữ liệu huấn luyện X và tập giá trị đúng (ground truth) Y . Mô hình mạng nơ-ron là một hàm $\mathcal{F} : (\theta, x) \rightarrow \hat{y}$ với x là một điểm dữ liệu và \hat{y} là giá trị mà mạng nơ-ron dự đoán ra dựa theo bộ trọng số θ . Như vậy, với tập dữ liệu huấn luyện X và tập giá trị đúng Y , chúng ta sẽ tính được độ lỗi của mô hình trên toàn tập dữ liệu huấn luyện với một hàm chi phí $\mathcal{L}(\theta)$ với θ là bộ trọng số của mô hình mạng nơ-ron:

$$E(\theta) = \frac{1}{N} \cdot \sum_{i=1}^N \mathcal{L}(\hat{y}_i, y_i) \quad (2.1)$$

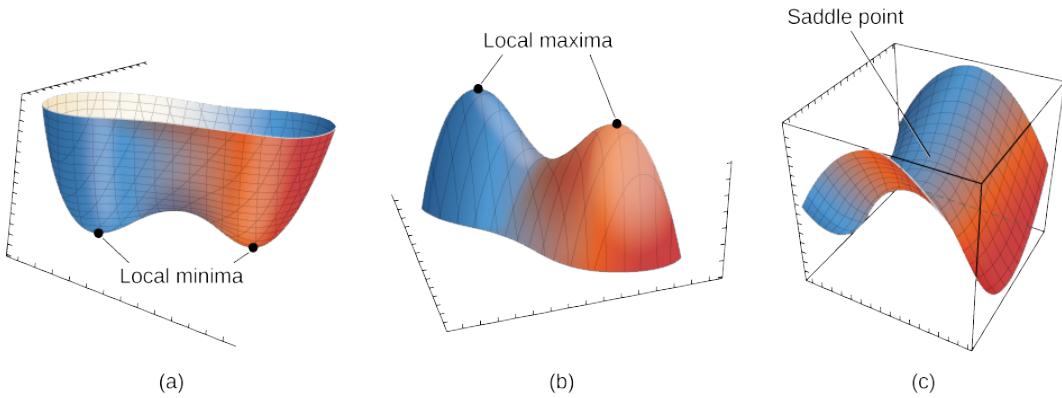
$$\Rightarrow E(\theta) = \frac{1}{N} \cdot \sum_{i=1}^N \mathcal{L}(\mathcal{F}_\theta(x_i), y_i) \quad (2.2)$$

Trong đó, θ là bộ trọng số của mô hình, $x_i = \{x_1, x_2, \dots, x_N\}$ là tập các tín hiệu đầu vào và N là kích thước tập dữ liệu huấn luyện, hàm $\mathcal{L}(\mathcal{F}_\theta(x_i), y_i)$ là hàm đánh giá độ lỗi của mạng nơ-ron với bộ trọng số θ và điểm dữ liệu x_i . Trung bình của các độ lỗi này là hàm chi phí $E(\theta)$ mà ta cần tối ưu. Với hàm chi phí $E(\theta)$ và không gian cao chiều được tạo bởi bộ trọng số, ta được một bề mặt phẳng lỗi M chiều với M là số trọng số của mạng nơ-ron.

Vì vậy, ta có thể đưa quá trình huấn luyện mạng nơ-ron thành quá trình đi tìm vị trí mà tại đó hàm chi phí $E(\theta)$ đạt giá trị cực tiểu. Giá trị này phải có độ lỗi rất nhỏ trên tập huấn luyện, đồng thời phải có độ lỗi đủ nhỏ trên dữ liệu ngoài tập huấn luyện để đảm bảo tính tổng quát hóa của mô hình. Tuy nhiên, việc đi tìm giá trị cực tiểu này gặp nhiều khó khăn do bộ trọng số của mô hình mạng nơ-ron nhiều tầng ẩn có số chiều rất lớn, cách giải phương trình thông thường không thể áp dụng tốt trong trường hợp này. Hơn nữa, mỗi trọng số có mức độ ảnh hưởng khác nhau

lên độ lỗi làm xuất hiện các điểm cực tiểu địa phương và điểm yên ngựa, các vùng bằng phẳng hay các vùng rãnh hẹp phức tạp.

“Critical point” (điểm tối hạn) là điểm mà tại đó hàm số không khả vi hoặc có đạo hàm bằng 0. Theo đó, cực tiểu địa phương của một hàm $f(x)$ là một điểm cực trị a trong một khoảng xác định có giá trị $f(a)$ không lớn hơn bất kỳ giá trị $f(x)$ với mọi x trong khoảng đó (Hình 2.5a). Công trình của D. Erhan và cộng sự[11] đã chứng minh rằng mạng nơ-ron nhiều tầng ẩn rất dễ hội tụ tại các điểm cực tiểu có độ lỗi khá cao nếu khởi tạo ngẫu nhiên. Các điểm này từng được coi là vấn đề chính trong việc hạn chế độ chính xác của mô hình và gây trở ngại trong việc ứng dụng các mô hình mạng nơ-ron nhiều tầng ẩn vào thực tế. Tuy nhiên một số nghiên cứu gần đây cho rằng điểm cực tiểu địa phương không phải là khó khăn chính cản trở quá trình huấn luyện của mạng nơ-ron nhiều tầng ẩn mà chính các điểm yên ngựa mới là trở ngại cho các thuật toán tối ưu[8].



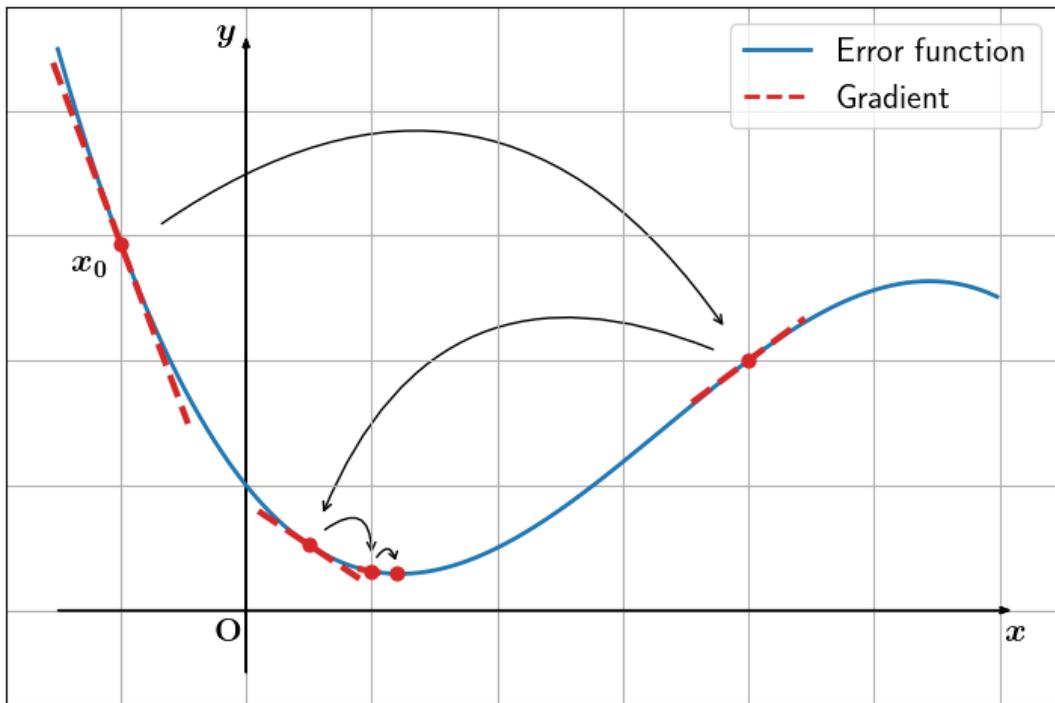
Hình 2.5: Minh họa điểm cực tiểu (a), cực đại (b) và điểm yên ngựa (c) trong không gian ba chiều.

Một hàm số $f(x, y)$ có điểm dừng tại (a, b) nhưng tại đó hàm số $f(x, y)$ không có cực trị, ta gọi điểm (a, b) là điểm yên ngựa (Hình 2.5c). Trong các mạng nơ-ron nông, điểm yên ngựa chiếm tỉ lệ rất ít trong số các critical point, đa số các critical point được tìm thấy là các cực tiểu địa phương. Tuy nhiên, Yann N. Dauphin đã chỉ ra rằng khi số lượng tầng ẩn trong mạng càng tăng thì tỉ lệ xuất hiện của các điểm yên ngựa là càng cao và gần như tất cả các critical point được tìm thấy là điểm yên ngựa có độ lỗi

cao hơn rất nhiều độ lỗi của cực tiểu toàn cục [8]. Đồng thời, bài báo cũng chỉ ra rằng trong không gian cao chiều, xác suất để tất cả các hướng xung quanh một critical point có đường cong lên là rất thấp, nghĩa là tần suất xuất hiện của cực tiểu địa phương là rất nhỏ. Vì những lý do trên mà các yên ngựa được coi là lý do lớn gây cản trở quá trình tối ưu mạng nơ-ron nhiều tầng ẩn.

Có hai khó khăn chính cần phải giải quyết khi gặp điểm yên ngựa: (1) Tìm được hướng có đường cong hướng xuống trong mặt phẳng lỗi để tiến tới điểm có độ lỗi nhỏ hơn, và (2) vượt qua vùng bằng phẳng, là vùng có độ dốc gần như bằng nhau, bao bọc xung quanh điểm yên ngựa. Trong các thuật toán tối ưu sử dụng đạo hàm bậc nhất, mặc dù hướng gradient trùng với hướng có đường cong hướng xuống trong mặt phẳng lỗi và có thể hướng tới điểm có độ lỗi nhỏ hơn, nhưng các bước cập nhật diễn ra rất chậm do độ dốc của các điểm này rất nhỏ và đường như bằng nhau trong vùng lân cận. Để khắc phục được khó khăn này, một số thuật toán bậc nhất đã cố gắng xấp xỉ thông tin đạo hàm bậc hai để có thêm thông tin về độ cong của mặt phẳng lỗi tại điểm đang xét. Các thuật toán này đều là những cải tiến của thuật toán Gradient Descent.

2.3 “Gradient Descent”



Hình 2.6: Minh họa quá trình tìm đến điểm cực tiểu của thuật toán Gradient Descent với đường màu xanh dương biểu diễn giá trị của hàm chi phí, đường đứt đoạn màu đỏ thể hiện phương và độ lớn của gradient.

“Gradient Descent” là thuật toán cho phép đi tìm cực tiểu của một hàm số mà không cần biết chính xác công thức của hàm số đó bằng cách di chuyển từng bước nhỏ cho đến khi hội tụ tại cực tiểu (Hình 2.6). Bằng cách sử dụng khai triển Taylor, ta có thể xấp xỉ đạo hàm bậc k của bất kỳ hàm số nào, cho phép Gradient Descent giải quyết được hầu hết các hàm số được dùng trong huấn luyện mạng nơ-ron nhiều tầng ẩn.

- Khai triển Taylor: Công thức 2.3 cho xấp xỉ đạo hàm bậc k của một hàm số $f(x)$ quanh một điểm x với Δx đủ nhỏ.

$$f(x+\Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2!}f''(x)(\Delta x)^2 + \dots + \frac{1}{k!}f^{(k)}(\Delta x)^{(k)} \quad (2.3)$$

Từ đó, để có thể cực tiểu hóa hàm chi phí \mathcal{L} , Gradient Descent sẽ di

chuyển theo hướng Δx sao cho $\mathcal{L}(x + \Delta x) < \mathcal{L}(x)$ và bằng:

$$\mathcal{L}(x + \Delta x) \approx \mathcal{L}(x) + \Delta x \nabla_x \mathcal{L} \quad (2.4)$$

với $\nabla_x \mathcal{L}$ là gradient của hàm chi phí \mathcal{L} tại x .

- Gradient là véc-tơ có hướng là hướng có mức độ tăng lớn nhất và độ lớn là mức độ thay đổi lớn nhất tại một điểm trong không gian. Giả sử một không gian \mathbb{R}^n được định nghĩa bởi một hàm $f(x_1, x_2, x_3, \dots, x_n)$ thì véc-tơ gradient tại một điểm trong không gian \mathbb{R}^n sẽ là một véc-tơ cột mà thành phần của nó là đạo hàm theo từng biến của hàm $f(x_1, x_2, x_3, \dots, x_n)$, cụ thể là:

$$\nabla f = \left(\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, \frac{\delta f}{\delta x_3}, \dots, \frac{\delta f}{\delta x_n} \right) \quad (2.5)$$

Vì gradient chỉ hướng có mức độ tăng lớn nhất nên hướng Δx cho giá trị của $\mathcal{L}(x + \Delta x)$ giảm nhiều nhất theo hướng $-\nabla_x \mathcal{L}$. Tuy nhiên, Δx phải có giá trị đủ nhỏ để xấp xỉ Taylor vẫn đảm bảo rằng hướng Δx sẽ cho giá trị $\mathcal{L}(x + \Delta x) < \mathcal{L}(x)$. Từ đó, ta áp dụng một hằng số dương η đảm bảo độ dài của một bước đi Δx đủ nhỏ, được gọi là tỷ lệ học (learning rate). Vậy ta có thể viết được công thức cập nhật của Gradient Descent tại mỗi bước nhảy x_t như sau:

$$\mathcal{L}(x_t + \Delta x) \approx \mathcal{L}(x_t) - \eta (\nabla_{x_t} \mathcal{L})^T \nabla_{x_t} \mathcal{L} \quad (2.6)$$

Lại có: $(\nabla_{x_t} \mathcal{L})^T \nabla_{x_t} \mathcal{L} > 0 \Rightarrow \mathcal{L}(x_t + \Delta x) < \mathcal{L}(x_t)$.

Với hằng số η cố định công thức 2.4 sẽ luôn tiến hành cập nhật một lượng $\Delta x = \eta (\nabla_{x_t} \mathcal{L})^T \nabla_{x_t} \mathcal{L} = \eta \|\nabla_{x_t} \mathcal{L}\|_2$ thay đổi tương ứng tỷ lệ với $\|\nabla_{x_t} \mathcal{L}\|_2$. Nếu $\|\nabla_{x_t} \mathcal{L}\|_2$ lớn, có nghĩa là độ dốc lớn, ta có thể bước một bước dài để tiến nhanh tới giá trị cực tiểu. Ngược lại, nếu $\|\nabla_{x_t} \mathcal{L}\|_2$ nhỏ, độ dốc nhỏ, ta phải bước những bước nhỏ để tránh bị vượt qua vùng cực tiểu.

Bên cạnh đó, nếu tỷ lệ học quá nhỏ, ta luôn bước những bước rất nhỏ hướng về phía cực tiểu nên sẽ mất rất nhiều thời gian để thuật toán có thể hội tụ. Tuy nhiên, nếu chọn một tỷ lệ học quá lớn thì xấp xỉ Taylor không đảm bảo được rằng giá trị $\mathcal{L}(x + \Delta x)$ sẽ giảm và thuật toán có khả năng không thể hội tụ. Đó là lý do tại sao lựa chọn một tỷ lệ học phù hợp là cần thiết.

2.3.1 “Batch” Gradient Descent

Thuật toán “Batch Gradient Descent” (BGD) là một phiên bản của thuật toán Gradient Descent sử dụng gradient của toàn bộ dữ liệu huấn luyện để thực hiện một bước cập nhật. Vì độ lỗi được tính tại các điểm dữ liệu riêng lẻ, nên trung bình các độ lỗi này sẽ là độ lỗi của cả tập dữ liệu (Công thức 2.2).

Từ công thức 2.2 và 2.6, ta có thể viết được công thức cập nhật của θ tại mỗi bước:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} E(\theta) \quad (2.7)$$

với $\nabla_{\theta} E(\theta)$ là gradient của hàm chi phí được tính trên toàn tập dữ liệu.

Tổng quan thuật toán BGD được trình bày trong thuật toán 1.

Thuật toán 1 Batch Gradient Descent (BGD)

Đầu vào: Tập dữ liệu huấn luyện x_i ($i = 1, 2, \dots, N$), tỉ lệ học η , bộ trọng số θ của mô hình \mathcal{F}

Đầu ra: Bộ trọng số θ của \mathcal{F} với độ lỗi đạt cực tiểu

Thao tác:

1: **while** θ chưa hội tụ **do**

2: Tính độ lỗi trung bình: $E(\theta) \leftarrow \frac{1}{N} \cdot \sum_{i=1}^N \mathcal{L}(\hat{y}_i, y_i)$

3: Thực hiện cập nhật trọng số: $\theta \leftarrow \theta - \eta \nabla_{\theta} E(\theta)$

4: **end while**

5: **return** θ

2.3.2 “Minibatch” Gradient Descent

Một trong những trở ngại của BGD khi ứng dụng vào việc huấn luyện mạng nơ-ron nhiều tầng ẩn là tốc độ huấn luyện. Việc huấn luyện một mạng nơ-ron nhiều tầng ẩn thường đòi hỏi một tập dữ liệu rất lớn để mô hình có thể học đủ thông tin về các đặc trưng của dữ liệu. Trong mỗi bước, BGD cần duyệt qua từng điểm dữ liệu trong tập dữ liệu huấn luyện, tính độ lỗi và véc-tơ đạo hàm riêng của độ lỗi theo từng trọng số, cuối cùng là tính trung bình của các véc-tơ đạo hàm để có được véc-tơ gradient $\nabla_{\theta} E(\theta)$, và chúng ta chỉ dùng một phần nhỏ của độ dài gradient đó để cập nhật.

Có thể thấy BGD mất rất nhiều thời gian để duyệt qua hết toàn bộ tập dữ liệu, nhưng chỉ cải thiện được một lượng rất nhỏ. Để khắc phục nhược điểm này, “Minibatch Gradient Descent” (MGD) chỉ thực hiện tính gradient của hàm chi phí trên một tập con của tập dữ liệu (thường gọi là một “minibatch”) để thực hiện một bước cập nhật.

Từ công thức 2.6 và 2.7 của BGD, chúng ta sẽ có công thức tính độ lỗi của MGD với một tập con:

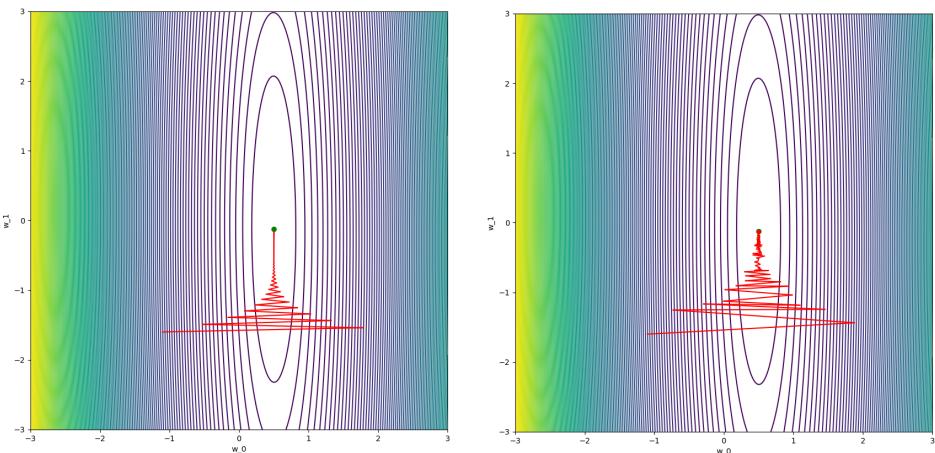
$$E(\theta) = \frac{1}{k} \cdot \sum_{i=1}^k \mathcal{L}(\mathcal{F}_{\theta}(x_i), y_i) \quad (2.8)$$

với k là kích thước của một tập con ($k << N$). Từ độ lỗi tính được, việc cập nhật trọng số cho mạng nơ-ron nhiều tầng ẩn được thực hiện tương tự như BGD bằng công thức 2.7.

Như vậy, có thể thấy được rằng một bước cập nhật với MGD sẽ nhanh hơn rất nhiều so với BGD do việc tính toán chỉ thực hiện trên k điểm dữ liệu và cập nhật theo gradient của tập con đó. Ngoài ra, trong mỗi lần duyệt qua toàn bộ tập dữ liệu (gọi là một “epoch”), MGD thực hiện được N/k lần cập nhật trọng số cho mô hình, thay vì chỉ một lần như BGD. Điều đó cũng đồng nghĩa rằng với cùng một số lượng epoch huấn luyện, MGD sẽ đi được rất nhiều bước so với BGD.

Một trường hợp đặc biệt của MGD là khi $k = 1$, được gọi là “Stochastic Gradient Descent” (SGD). Tuy nhiên hiện nay khi nhắc đến SGD trong bài toán tối ưu mạng nơ-ron nhiều tầng ẩn, người ta thường ám chỉ MGD với $k > 1$, từ “Stochastic” ám chỉ cách chọn các tập con từ tập dữ liệu huấn luyện là ngẫu nhiên. Việc sử dụng khái niệm “Batch” trong BGD cũng dễ gây nhầm lẫn vì từ “Batch” mang ý nghĩa một tập hợp của thứ gì đó, như “batch size” là kích thước của một tập con, nhưng ở đây lại ám chỉ cả tập dữ liệu huấn luyện[12].

Trong đa số các bài báo khoa học, các tác giả sử dụng khái niệm “Gradient Descent” (GD) để chỉ BGD, và “Stochastic Gradient Descent” để nói tới MGD. Để tạo sự thống nhất cũng như thuận tiện trong việc liên hệ giữa nội dung khoa luận với nội dung của các bài báo khoa học, từ thời điểm này, chúng tôi cũng sẽ sử dụng cách gọi tên tương tự cho các thuật toán này.



Hình 2.7: So sánh đường đi của Gradient Descent (trái) và Stochastic Gradient Descent (phải).

Việc chỉ sử dụng một tập con của dữ liệu huấn luyện để tính gradient cũng đồng nghĩa với việc hướng của gradient này sẽ chỉ xấp xỉ gradient tính được trên cả tập dữ liệu (có thể gọi là “true gradient”), và sẽ có những sự dao động. Tuy nhiên, nhiều nghiên cứu đã chỉ ra rằng các véc-tơ gradient

của các tập con sẽ dao động xung quanh hướng của véc-tơ gradient tính được theo GD với độ chênh lệch không quá lớn[5][6]. Việc lựa chọn kích thước tập con k là sự đánh đổi giữa mức độ dao động với tốc độ tính toán cũng như số bước cập nhật được thực hiện trong mỗi epoch. Khi k tiến tới gần N , hướng gradient của tập con sẽ xấp xỉ hướng của true gradient tốt hơn, nhưng lại mất nhiều thời gian tính toán hơn. Ngược lại, khi k tiến gần về 1, tuy thời gian tính toán được rút ngắn nhưng hướng gradient của tập con càng chênh lệch nhiều so với hướng của true gradient, gây ra sự nhiễu loạn.

Tổng quan thuật toán SGD được trình bày trong thuật toán 2.

Thuật toán 2 Stochastic Gradient Descent (BGD)

Đầu vào: Tập dữ liệu huấn luyện x_i ($i = 1, 2, \dots, N$), kích thước tập con k , tỉ lệ học η , bộ trọng số θ của mô hình \mathcal{F}

Đầu ra: Bộ trọng số θ của \mathcal{F} với độ lỗi đạt cực tiểu

Thao tác:

- 1: **while** θ chưa hội tụ **do**
- 2: Xáo trộn tập dữ liệu.
- 3: **for** mỗi tập con kích thước k **do**
- 4: Tính độ lỗi trung bình: $E(\theta) \leftarrow \frac{1}{k} \cdot \sum_{i=1}^k \mathcal{L}(\hat{y}_i, y_i)$
- 5: Thực hiện cập nhật trọng số: $\theta \leftarrow \theta - \eta \nabla_{\theta} E(\theta)$
- 6: **end for**
- 7: **end while**
- 8: return θ

2.4 “Lan truyền ngược” (Backpropagation)

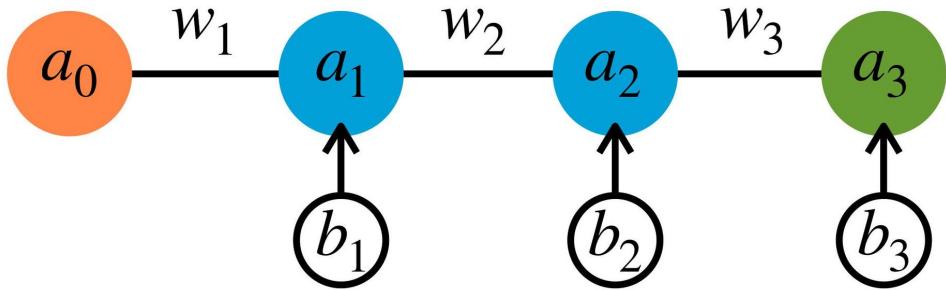
Quá trình huấn luyện mạng nơ-ron cần tìm một bộ trọng số cho độ lỗi là nhỏ nhất và việc tìm kiếm này có thể được hoàn thành bằng các thuật toán tối ưu đã trình bày ở trên. Tuy nhiên các thuật toán tối ưu này cần véc-tơ đạo hàm riêng của hàm chi phí tại từng trọng số, có nghĩa là ta muốn tìm sự phụ thuộc của độ lỗi và các giá trị trọng số trong mạng. Thuật toán lan

truyền ngược có thể giải quyết được vấn đề này bằng cách “lan truyền” độ lỗi trở lại trong mạng, từ đó có thể thay đổi giá trị của các trọng số tương ứng với mức độ ảnh hưởng lên độ lỗi. Ý tưởng cho thuật toán được giới thiệu trong bài báo của nhóm tác giả David Rumelhart, Geoffrey Hinton và Ronald Williams dưới cái tên Generalized Delta Rule[22]. Thuật toán ra đời đã đánh dấu một sự phát triển vượt bậc trong việc huấn luyện mạng nơ-ron nhiều tầng ẩn và nhận được sự quan tâm rất lớn trong cộng đồng nghiên cứu.

Thuật toán lan truyền ngược là một phép vi phân ngược (reverse-mode differentiation) sử dụng quy tắc mắt xích (chain rule) để tính toán mức độ ảnh hưởng của một trọng số lên giá trị độ lỗi của mạng nơ-ron nhiều tầng ẩn và được thể hiện qua giá trị độ lớn của các phần tử trong véc-tơ $\nabla_{\theta}E(\theta)$ tương ứng cho từng trọng số. Nếu một phần tử trong véc-tơ $\nabla_{\theta}E(\theta)$ có giá trị lớn, nghĩa là khi thay đổi trọng số tương ứng một lượng nhỏ thì độ lỗi sẽ thay đổi một lượng lớn, và ngược lại, nếu một phần tử trong $\nabla_{\theta}E(\theta)$ có giá trị nhỏ bị thay đổi thì độ lỗi sẽ chỉ biến thiên một lượng nhỏ. Vì tính chất đó mà ta gọi giá trị này là “độ nhạy cảm” của trọng số.

“Độ nhạy cảm” của một trọng số $w_i \in \theta$ được tính bằng tỉ lệ giữa độ biến thiên của trọng số đó và độ biến thiên của hàm chi phí $E(\theta)$ và bằng đạo hàm riêng của hàm $E(\theta)$ tại từng trọng số: $\frac{\delta E}{\delta w_1}; \frac{\delta E}{\delta w_2}; \dots; \frac{\delta E}{\delta w_n}$. Tổng hợp của các giá trị này cho ta véc-tơ $\nabla_{\theta}E(\theta)$. Để dễ dàng trong việc giải thích cách thuật toán thực thi, ta sử dụng một mạng nơ-ron nhiều tầng ẩn đơn giản, được mô tả như Hình 2.8 với 1 tầng nhập, 2 tầng ẩn và 1 tầng xuất, mỗi tầng chỉ có một nơ-ron.

Khi ta truyền tín hiệu đầu vào a_0 vào mô hình, sau khi rút trích đặc trưng qua 2 tầng ẩn, ta được kết quả dự đoán ở a_3 . Đem so sánh kết quả tại a_3 với giá trị nhãn đúng y ta được độ lỗi $E(\theta)$ được tính bằng công thức sau:



Hình 2.8: Minh họa mô hình mạng nơ-ron đơn giản gồm ba tầng: 1 tầng nhập (màu cam), 2 tầng ẩn (màu lam) và 1 tầng xuất (màu lục). Trong đó, w_i là các trọng số liên kết, b_i là hệ số bias tương ứng của từng nơ-ron và a_i là giá trị kích hoạt của nơ-ron.

$$\begin{cases} E(\theta) = (a_3 - y)^2 \\ a_3 = \sigma(w_3 \cdot a_2 + b_3) \end{cases} \rightarrow E(\theta) = (\sigma(w_3 \cdot a_2 + b_3) - y)^2 \quad (2.9)$$

Từ độ lỗi này, ta cần tìm đạo hàm riêng của $E(\theta)$ theo từng trọng số. Theo quy tắc mắt xích, ta có:

$$\begin{aligned} \frac{\delta E}{\delta w_3} &= \frac{\delta E}{\delta \sigma} \cdot \frac{\delta \sigma}{\delta(w_3 a_2 + b_3)} \cdot \frac{\delta(w_3 a_2 + b_3)}{\delta w_3} \\ &= 2(\sigma(w_3 a_2 + b_3) - y) \cdot \sigma(w_3 a_2 + b_3)' \cdot a_2 \end{aligned} \quad (2.10)$$

với $\sigma(x)$ là một hàm kích hoạt bất kỳ. Đặt $z_3 = \theta_3 a_2 + b_3$, ta viết lại công thức 2.10:

$$\frac{\delta E}{\delta w_3} = 2(\sigma(z_3) - y) \cdot \sigma(z_3) \cdot a_2 \quad (2.11)$$

Tương tự như vậy, ta có đạo hàm riêng của $E(\theta)$ tại w_1 và w_2 :

$$\begin{aligned}\frac{\delta E}{\delta w_2} &= \frac{\delta E}{\delta \sigma(z_2)} \cdot \frac{\delta \sigma(z_2)}{\delta z_2} \cdot \frac{\delta z_2}{\delta w_2} \\ &= 2(\sigma(z_3) - y) \cdot \sigma(z_3)' \cdot w_3 \cdot \sigma(z_2)' \cdot a_1\end{aligned}\quad (2.12)$$

$$\begin{aligned}\frac{\delta E}{\delta w_1} &= \frac{\delta E}{\delta \sigma(z_1)} \cdot \frac{\delta \sigma(z_1)}{\delta z_1} \cdot \frac{\delta z_1}{\delta w_1} \\ &= 2(\sigma(z_3) - y) \cdot \sigma(z_3)' \cdot w_3 \cdot \sigma(z_2)' \cdot w_2 \cdot \sigma(z_1)' \cdot a_0\end{aligned}\quad (2.13)$$

Các công thức 2.11, 2.12 và 2.13 được truyền qua tập dữ liệu và lấy giá trị trung bình. Tổng hợp các giá trị này ta được véc-tơ đạo hàm riêng $\nabla_{\theta}E(\theta)$ được dùng trong các thuật toán tối ưu khác. Tổng quát hoá cho một mô hình có L tầng và số lượng nơ-ron ở mỗi tầng là $n^{(l)}$ thì độ phụ thuộc của giá trị độ lỗi vào trọng số liên kết giữa nơ-ron thứ k của tầng thứ $l-1$ và nơ-ron thứ j của tầng thứ l là w_{jk} , với $z_j = \dots + w_{jk}a_k^{l-1} + \dots + b_j$, ta có:

$$\frac{\delta E}{\delta w_{jk}} = \frac{\delta E}{\delta \sigma(z_j)} \cdot \frac{\delta \sigma(z_j)}{\delta z_j} \cdot \frac{\delta z_j}{\delta w_{jk}} \quad (2.14)$$

và đạo hàm riêng của độ lỗi tại nơ-ron thứ k của tầng thứ $l-1$ là giá trị tổng hợp của các hàm kích hoạt của tầng thứ l .

$$\frac{\delta E}{\delta a_k^{l-1}} = \sum_{j=0}^{n^{l-1}} \frac{\delta E}{\delta \sigma(z_j)} \cdot \frac{\delta \sigma(z_j)}{\delta z_j} \cdot \frac{\delta z_j}{\delta a_k^{l-1}} \quad (2.15)$$

Từ các công thức ta có thể nhận thấy rằng đây là quá trình cho một độ lỗi nhất định, không có công thức chung cho tất cả giá trị độ lỗi và toàn bộ quá trình sẽ phải được lặp lại khi có một độ lỗi mới. Đây là một điểm yếu của phương pháp vi phân ngược nhưng lại rất thích hợp trong huấn luyện mạng nơ-ron nhiều tầng ẩn.

Chương 3

Huấn luyện mạng nơ-ron nhiều tầng ẩn bằng thuật toán Adam

Chương này trình bày về thuật toán tối ưu Adam và cách thuật toán khắc phục khó khăn trong huấn luyện mạng nơ-ron nhiều tầng ẩn mà khoá luận tập trung tìm hiểu. Đầu tiên chúng tôi trình bày về các thuật toán nền tảng: (1) thuật toán Gradient Descent với Momentum để tăng tốc và giảm dao động trong quá trình di chuyển trên bề mặt lỗi, và (2) thuật toán Gradient Descent với tỉ lệ học thích ứng với từng trọng số, cụ thể là thuật toán Adagrad và RMSprop. Dựa trên nền hai thuật toán này, chúng tôi trình bày ý tưởng của thuật toán Adam cũng như những ưu/khuyết điểm của thuật toán trong giải quyết các khó khăn của bài toán huấn luyện mạng nơ-ron.

3.1 Thuật toán Gradient Descent với Momentum

Các thuật toán sử dụng gradient như GD và SGD gặp phải hai khó khăn chính khi chỉ có thông tin của véc-tơ gradient mà thiếu đi thông tin về độ cong của bề mặt lỗi. Đầu tiên, vì hướng của gradient là hướng có độ dốc lớn nhất nên không phải lúc nào cũng là hướng tốt nhất và có thể gây dao động mạnh trong khi độ lỗi giảm không nhiều. Thứ hai, “độ nhạy cảm” của mỗi trọng số là khác nhau trong bề mặt lỗi, vì vậy áp dụng một tỷ lệ học chung cho tất cả các trọng số sẽ khiến cho hướng cập nhật bỏ qua các trọng số có “độ nhạy cảm” nhỏ.

“GD/SGD với Momentum” (hay “Momentum”) đều được dùng để chỉ một phương pháp cải tiến sử dụng quán tính để điều chỉnh hướng gradient theo hướng tốt nhất về cực tiểu, từ đó tăng tốc độ di chuyển trên bề mặt lỗi. Quán tính được thêm vào công thức thông qua hệ số quán tính β được chọn trước khi trình huấn luyện và công thức cập nhật θ tại mỗi bước được cải tiến như công thức 3.2. Dưới góc nhìn vật lý, Momentum trong công thức 3.1 là phần vận tốc còn lại sau khi tiêu hao trong quá trình vật di chuyển về vị trí cân bằng, quyết định mức độ dao động và tốc độ di chuyển của vật. Với một hệ số momentum càng cao, lượng vận tốc tiêu hao càng ít, vận tốc càng lớn thì lượng dao động của vật càng nhiều khiến cho vật mất nhiều thời gian để trở về vị trí cân bằng. Ngược lại, nếu hệ số momentum quá thấp, lượng vận tốc tiêu hao quá lớn dẫn đến vật không thể đến vị trí cân bằng. Tương tự như vậy, trong huấn luyện mạng nơ-ron, một hệ số momentum quá cao sẽ khiến cho hướng di chuyển bị dao động nhiều quanh điểm cực tiểu nhưng không thể hội tụ, và ngược lại, thời gian huấn luyện sẽ tăng lên đáng kể.

$$v_t = \beta v_{t-1} + \alpha \nabla E(\theta_t) \quad (3.1)$$

$$\theta_t = \theta_{t-1} - v_t \quad (3.2)$$

Có thể nói hệ số momentum điều chỉnh mức độ “ghi nhớ” của thuật toán. Hệ số momentum càng lớn thì giá trị quan tính cũ được ghi nhớ càng lâu và ngược lại. Điều đó liên quan đến đường trung bình động hàm mũ (Exponential Moving Averages) hay EMA. Đường trung bình động hàm mũ cho phép ta sử dụng trung bình động để xử lý nhiễu trong dữ liệu mà ta quan sát được và xấp xỉ nó gần hơn với dữ liệu thực. Ta thực hiện lấy trung bình trên $\frac{1}{1-\beta}$ bước nhảy gần nhất và thực hiện cập nhật trọng số. Ta khai triển công thức 3.1 như sau:

$$\begin{aligned} v_1 &= \beta v_0 + \alpha \nabla E(\theta_1) \\ v_2 &= \beta v_1 + \alpha \nabla E(\theta_2) = \beta(v_0 + \alpha \nabla E(\theta_1)) + \alpha \nabla E(\theta_2) \\ &= \beta v_0 + \beta \alpha \nabla E(\theta_1) + \alpha \nabla E(\theta_2) \\ v_3 &= \beta v_2 + \alpha \nabla E(\theta_3) = \beta(\beta v_0 + \beta \alpha \nabla E(\theta_1) + \alpha \nabla E(\theta_2)) + \alpha \nabla E(\theta_3) \\ &= \beta^2 v_0 + \beta^2 \alpha \nabla E(\theta_1) + \beta \alpha \nabla E(\theta_2) + \alpha \nabla E(\theta_3) \end{aligned} \quad (3.3)$$

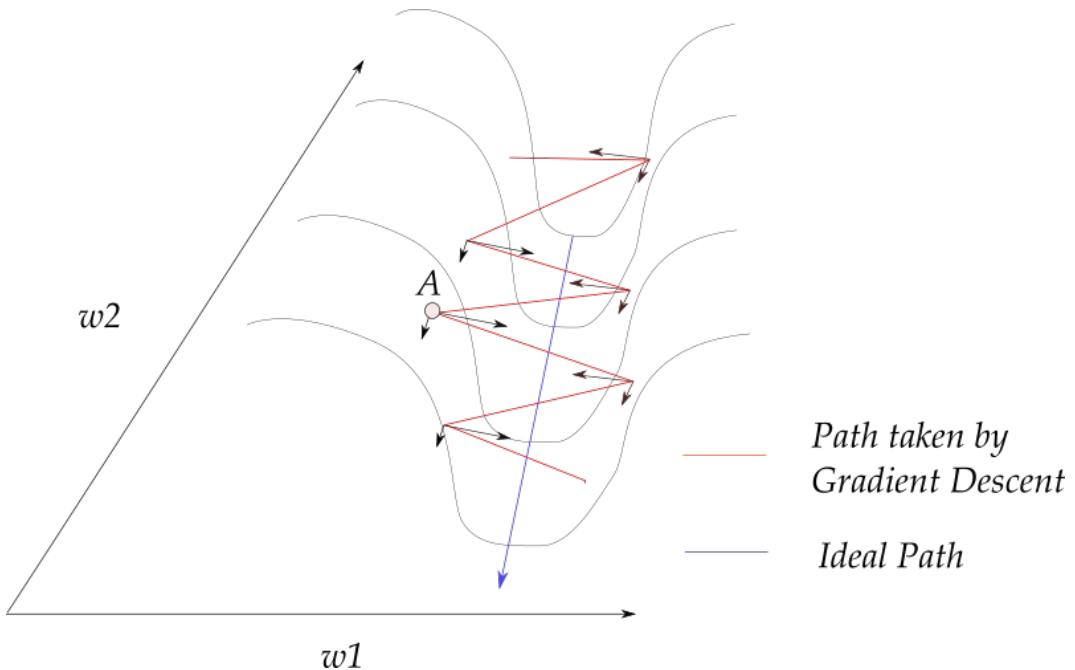
Từ công thức 3.3 ta nhận thấy rằng tại bước nhảy thứ t , giá trị của momentum phụ thuộc vào tất cả các giá trị momentum trước đó từ $1..(t-1)$. Mỗi giá trị trong dãy đều được nhân với hệ số β^t . Vì β là hệ số quan tính nằm trong khoảng $(0, 1)$ nên β^t sẽ càng nhỏ khi t càng lớn dẫn đến các giá trị càng lâu trước đó sẽ có hệ số càng nhỏ và dần không đóng góp gì nhiều trong quá trình huấn luyện. Hay nói cách khác, các giá trị này dần được quên đi và thuật toán Momentum quan tâm nhiều đến các giá trị gần với hiện tại.

Trong thực tế huấn luyện, ta thường sử dụng cách xấp xỉ true gradient bằng các gradient của tập con để tăng tốc quá trình cập nhật trọng số. Trong quá trình xấp xỉ, việc tạo minibatch bằng cách gom nhóm ngẫu nhiên các điểm dữ liệu đã thêm một lượng nhiễu vào trong quá trình tối ưu. So sánh độ lớn của nhiễu với độ lớn của momentum ta có thể chia quá

trình thành hai giai đoạn: giai đoạn “transient” và giai đoạn “fine-tuning”. Trong giai đoạn transient, momentum vẫn còn hiệu quả trong điều chỉnh hướng của gradient và giúp gradient tiến nhanh về vị trí có cực tiểu vì độ lớn của momentum lớn hơn độ lớn của nhiễu. Quá trình huấn luyện càng lâu, độ lớn của momentum ngày càng giảm và bị lấn át bởi độ nhiễu, khi đó ta bước qua giai đoạn fine-tuning. Trong các mạng nơ-ron nhiều tầng ẩn, giai đoạn fine-tuning chiếm một phần ít hơn và cũng ít quan trọng hơn giai đoạn transient nên việc đảm bảo giai đoạn transient được diễn ra đủ lâu là một sự cần thiết [27].

Sự chuyển đổi giữa hai quá trình được kể trên được quyết định bởi hằng số quán tính β . Sử dụng một hệ số quán tính β lớn cho phép momentum thực hiện quá trình tối ưu tốt hơn trên các hướng có “độ nhạy cảm” nhỏ. Tuy nhiên, lợi thế này không được thể hiện qua rõ ràng qua độ lỗi do thuật toán không thể hội tụ tốt tại các hướng có “độ nhạy cảm” cao với một hệ số quán tính β lớn. Sự đánh đổi này cho kết quả là ta có thể tiến gần đến vùng có cực tiểu hoặc là hội tụ tại cực tiểu có độ lỗi nhỏ hơn. Trong khi sử dụng một hệ số β nhỏ sẽ cho độ lỗi cao hơn vì giai đoạn transient kết thúc quá sớm. Thiếu đi sự hỗ trợ của momentum, các phương pháp bậc nhất không thể tận dụng tốt thông tin tại các hướng có “độ nhạy cảm” nhỏ và rất dễ bỏ qua các vùng chứa cực tiểu hoặc cực tiểu có độ lỗi thấp hơn dẫn đến hội tụ tại điểm có độ lỗi cao.

Đặc biệt trong trường hợp rãnh hẹp, momentum giúp GD/SGD tăng tốc độ hội tụ đáng kể. Ta có thể thấy trong Hình 3.1, tại một điểm A bất kỳ, ta có thể phân tích thành hai véc-tơ thành phần của từng trọng số. Vì rãnh hẹp nên sẽ có một hướng có độ dốc cao hơn hướng còn lại, trong trường hợp này là w_1 , nên độ dài véc-tơ tương ứng tại hướng đó sẽ có độ dài lớn hơn độ dài véc-tơ tại hướng còn lại. Thuật toán GD/SGD sẽ ưu tiên hướng có độ dốc cao hơn và bỏ qua hướng w_2 có độ dốc thấp, gây ra hiện tượng dao động cho độ lỗi giảm ít với thời gian huấn luyện lâu. Một giải pháp thường được sử dụng để khắc phục hiện tượng này là giảm tỉ lệ học của thuật toán GD/SGD. Tuy nhiên, để khắc phục hiện tượng dao



Hình 3.1: Minh họa cách Momentum triệt tiêu hướng có độ dốc lớn. (Nguồn: <https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/>)

động, ta cần một tỉ lệ học rất nhỏ, với tỉ lệ học này thuật toán GD/SGD sẽ không thể “khám phá” những hướng có độ dốc nhỏ một cách hiệu quả do tốc độ giá trị gradient biến đổi tại những hướng này là rất chậm và dễ dàng bị lấn át bởi hướng có độ dốc cao. Trong trường hợp hướng đi về cực tiểu là một trong các hướng có độ dốc thấp, thuật toán GD/SGD sẽ thực hiện những bước cập nhật rất nhỏ, cho cảm giác huấn luyện bị chững lại và tạo cảm giác gấp phải cực tiểu địa phương.

Momentum giải quyết vấn đề này bằng cách cộng dồn các véc-tơ quán tính. Khi cộng dồn, các véc-tơ tại hướng có độ dốc lớn w_1 sẽ bị triệt tiêu và véc-tơ tại hướng có độ dốc nhỏ w_2 được tăng cường. Từ đó, momentum giúp điều chỉnh hướng của gradient và tăng tốc độ hội tụ của thuật toán.

3.2 Thuật toán Gradient Descent với tỉ lệ học thích ứng

Việc dự đoán trong học máy nói chung và mạng nơ-ron nhiều tầng ẩn nói riêng dựa vào quá trình trích xuất và liên hệ các đặc trưng của dữ liệu. Thông thường, các điểm dữ liệu trong cùng một bài toán sẽ có những đặc trưng chung thường gặp. Tuy nhiên, cũng có một số đặc trưng rất hiếm khi xuất hiện, chỉ có ở trong một số ít điểm dữ liệu cụ thể. Các đặc trưng hiếm gặp này thường sẽ có ảnh hưởng rất lớn về mặt ý nghĩa của dữ liệu, và mang lại rất nhiều thông tin về điểm dữ liệu đó[23]. Một ví dụ cụ thể là trong phân tích văn bản, các từ xuất hiện thường xuyên như các liên từ (“và”, “thì”, “nhưng”,...) hay các phó từ (“cũng”, “lại”, “rồi”,...) lại không thể hiện nội dung nhiều bằng các danh từ và động từ chỉ các đối tượng, hành động cụ thể. Một trường hợp khác là khi xét riêng một đặc trưng, những điểm dữ liệu mang giá trị khác với các điểm dữ liệu còn lại đóng vai trò quan trọng hơn trong việc cung cấp thông tin để giải quyết bài toán.

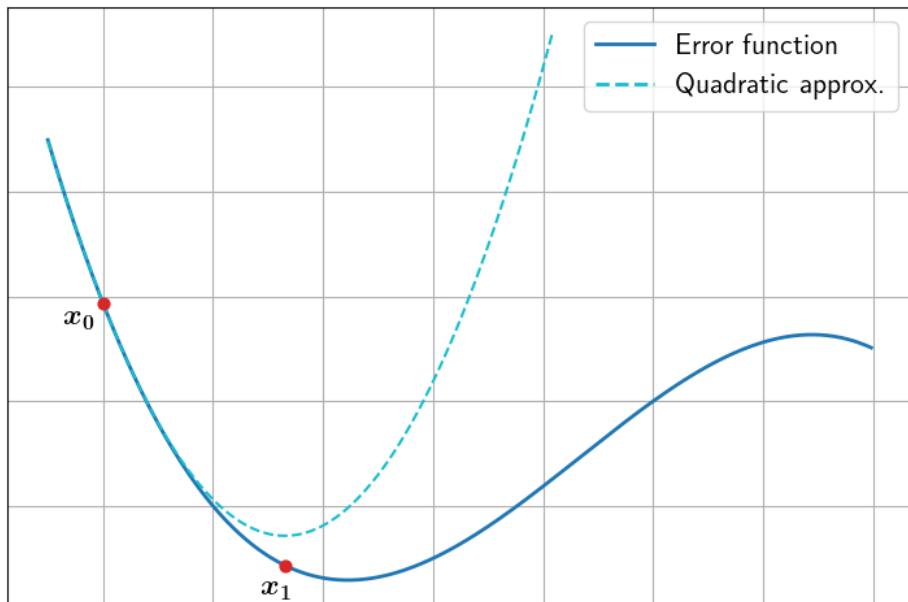
Tuy nhiên, các thông tin quan trọng được biểu diễn bằng các đặc trưng hiếm lại ít được mạng nơ-ron nhiều tầng ẩn chú ý. Trong mạng nơ-ron nhiều tầng ẩn, các nơ-ron tương ứng với các đặc trưng thường gặp sẽ được cập nhật thường xuyên, trong khi các nơ-ron tương ứng với các đặc trưng hiếm chỉ được cập nhật một số ít lần trong một lần duyệt qua toàn bộ dữ liệu. Hiện tượng này khiến cho các đặc trưng thường gặp và mang ít thông tin được biểu diễn tốt hơn, còn các đặc trưng hiếm mang nhiều ý nghĩa lại chưa được học.

Mặc dù SGD và Momentum thực hiện cập nhật một lượng khác nhau cho mỗi trọng số tùy theo độ lớn của gradient tại điểm đang xét (và trạng thái của momentum tại thời điểm đó), tuy nhiên lượng cập nhật vẫn chỉ dựa vào hướng và độ dốc của gradient, hay nói cách khác là độ dốc của bề mặt lỗi, tại điểm đang xét và chưa tính tới độ nhạy cảm của các trọng số. John Duchi, Elad Hazan, và Yoram Singer đã đề xuất thuật toán

Adagrad[10] lấy ý tưởng từ phương pháp Newton để giải quyết vấn đề này bằng cách thích ứng lượng cập nhật cho từng trọng số: các trọng số tương ứng với các đặc trưng thường gấp sẽ được cập nhật ít hơn, còn các trọng số tương ứng với các đặc trưng hiếm sẽ được cập nhật lượng lớn hơn.

- Từ khai triển Taylor ở công thức 2.3, chúng ta có thể tạo được một xấp xỉ bậc hai ρ của hàm f cho các giá trị x xung quanh một điểm x_0 :

$$f(x) \approx \rho(x) = f(x_0) + f'(x_0) \cdot (x - x_0) + \frac{1}{2} \cdot f''(x_0) \cdot (x - x_0)^2 \quad (3.4)$$



Hình 3.2: Xấp xỉ bậc hai (đường đứt khúc màu xanh nhạt) của hàm lỗi (đường màu xanh đậm) và bước cập nhật tiếp theo của phương pháp Newton.

Vì đây là một xấp xỉ bậc hai nên chúng ta có thể tìm cực trị của xấp xỉ này bằng cách giải phương trình $\rho'(x) = 0$. Khi đó nghiệm của $\rho'(x)$ sẽ là bước cập nhật tiếp theo của phương pháp Newton

(hình 3.2) được tính bằng công thức 3.5:

$$\begin{aligned}
 \rho'(x_{t+1}) &= 0 \\
 \Rightarrow f'(x_t) + f''(x_t) \cdot (x_{t+1} - x_t) &= 0 \\
 \Rightarrow x_{t+1} &= x_t - \frac{f'(x_t)}{f''(x_t)}
 \end{aligned} \tag{3.5}$$

- Ma trận Hessian là ma trận vuông gồm các đạo hàm bậc hai theo từng cặp hướng của một hàm số. Cho một hàm số f có tham số đầu vào là một véc-tơ $x \in \mathbb{R}^M$, chúng ta sẽ có ma trận vuông Hessian H kích thước $M \times M$ theo công thức:

$$H = \begin{bmatrix} \frac{\delta^2 f}{\delta x_1^2} & \frac{\delta^2 f}{\delta x_1 \delta x_2} & \cdots & \frac{\delta^2 f}{\delta x_1 \delta x_M} \\ \frac{\delta^2 f}{\delta x_2 \delta x_1} & \frac{\delta^2 f}{\delta x_2^2} & \cdots & \frac{\delta^2 f}{\delta x_2 \delta x_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta^2 f}{\delta x_M \delta x_1} & \frac{\delta^2 f}{\delta x_M \delta x_2} & \cdots & \frac{\delta^2 f}{\delta x_M^2} \end{bmatrix} \tag{3.6}$$

- Để khái quát hóa phương pháp Newton cho bài toán tối ưu trong không gian cao chiều, ta thay đạo hàm bậc nhất và đạo hàm bậc hai từ công thức 3.4 lần lượt bằng gradient $g(x_0)$ và ma trận Hessian H tại điểm x_0 :

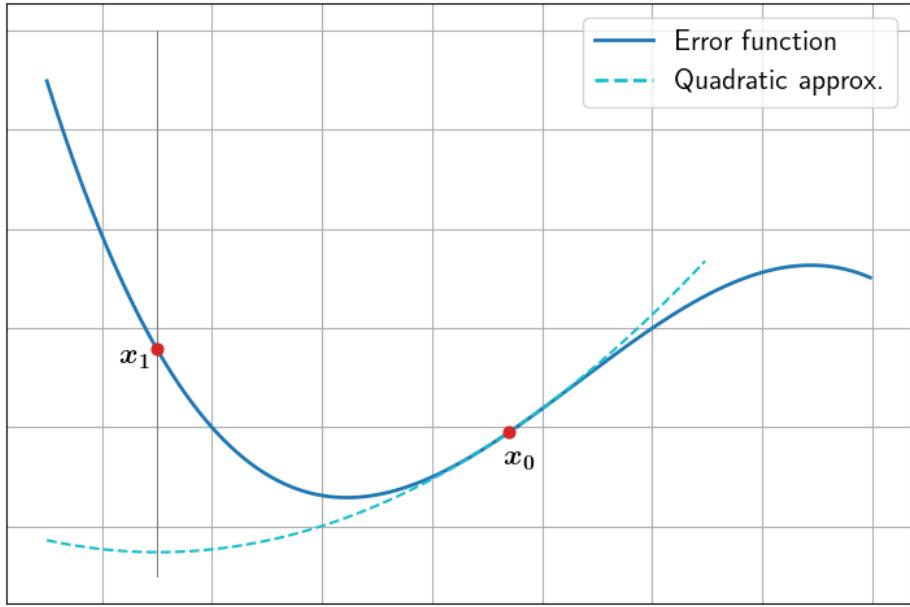
$$f(x) \approx \rho(x) = f(x_0) + g(x_0)^\top (x - x_0) + \frac{1}{2}(x - x_0)^\top H(x - x_0) \tag{3.7}$$

Tương tự như trên, chúng ta cũng giải phương trình $\nabla \rho(x) = 0$ để tìm bước cập nhật tiếp theo:

$$\begin{aligned}
 \nabla \rho(x_t) &= 0 \\
 \Rightarrow g(x_t) + H_t \cdot (x_{t+1} - x_t) &= 0 \\
 \Rightarrow x_{t+1} &= x_t - H_t^{-1} \cdot g(x_t)
 \end{aligned} \tag{3.8}$$

Tại mỗi bước cập nhật, phương pháp Newton xấp xỉ bề mặt lỗi xung quanh điểm hiện tại bằng một parabol P có cùng độ dốc và độ cong với bề mặt lỗi tại điểm đang xét. Từ đó, bước cập nhật trong công thức 3.8 sẽ cho một điểm mới là điểm cực trị trong parabol P (cực tiểu nếu parabol P có bề lõm quay lên trên; cực tiểu nếu parabol P có bề lõm quay xuống dưới). Ví dụ một hàm số bậc hai có dạng $y_1 = ax^2 + bx + c$ sẽ có đồ thị là một parabol P_1 có bề lõm hướng lên và điểm cực tiểu là điểm $A(\frac{-b}{2a}; \frac{-\Delta}{4a})$. Lấy một điểm B bất kỳ thuộc parabol P_1 có tọa độ (x_B, y_B) với $x_B > \frac{-b}{2a}$. Để từ B đến được cực tiểu A trong một bước, thì tạo độ điểm B phải thay đổi một lượng $\Delta x = x_B - x_A = x_B + \frac{b}{2a} = \frac{2ax_B + b}{2a} = \frac{f'(x_B)}{f''(x_B)}$. Vậy từ điểm B ta thực hiện bước cập nhật $x_B - \Delta x$ thì ta đến được điểm A . Trong trường hợp hàm nhiều biến, đạo hàm bậc nhất $f'(x_B)$ sẽ được thay bằng véc-tơ gradient và đạo hàm bậc hai sẽ được thay bằng ma trận Hessian. Viết lại theo dạng tổng quát ta sẽ được công thức 3.8. Vì vậy các thuật toán bậc hai bị "thu hút" bởi các điểm cực trị dẫn đến mắc kẹt tại điểm yên ngựa. Đồng thời đây là một trong những lý do quan trọng khiến các phương pháp bậc hai ít được sử dụng hơn các phương pháp bậc nhất trong huấn luyện mạng nơ-ron nhiều tầng ẩn. Các phương pháp bậc nhất không gặp vấn đề này do hướng của gradient là hướng có độ thay đổi lớn nhất, đồng nghĩa với việc hướng gradient luôn chỉ về cực đại. Bằng cách đi ngược lại với hướng của gradient, ta luôn có thể tìm về một điểm mới có độ lỗi nhỏ hơn điểm hiện tại. Vấn đề của các phương pháp bậc nhất gặp phải là độ lớn bước cập nhật quá nhỏ. Các phương pháp tỉ lệ học thích ứng giúp thay đổi điều này bằng thông tin xấp xỉ được từ ma trận Hessian để thay đổi kích thước của bước cập nhật.

Ngoài ra, phương pháp Newton phụ thuộc khá lớn vào việc xấp xỉ cục bộ tại một điểm trên bề mặt lỗi. Vì vậy, một xấp xỉ không chính xác dễ dàng gây ra những bước cập nhật kém hiệu quả và có khả năng thuật toán không thể hội tụ tại cực tiểu. Các trường hợp này thường xảy ra khi các đạo hàm riêng bậc hai bằng 0 hoặc tiến gần về 0, khiến cho việc xấp xỉ bằng ma trận nghịch đảo không còn chính xác. Từ đó công thức 3.8 cho



Hình 3.3: Cực tiểu của xấp xỉ bậc hai cung cấp một bước cập nhật không tối ưu.

ta một điểm mới có độ lỗi lớn hơn điểm hiện tại (hình 3.3).

Một hạn chế khác của các phương pháp bậc hai như phương pháp Newton nằm ở việc tính toán ma trận Hessian. Trong không gian \mathbb{R}^M với M có thể lên tới hàng trăm triệu hay thậm chí là hàng tỷ tương ứng với số trọng số của mô hình, ma trận Hessian sẽ yêu cầu thời gian tính toán và bộ nhớ quá lớn, dẫn đến việc sử dụng các phương pháp bậc hai cho bài toán tối ưu mạng nơ-ron nhiều tầng ẩn là bất khả thi.

Thuật toán Adagrad cũng sử dụng nguyên lý xấp xỉ bề mặt lỗi tương tự như phương pháp Newton. Tuy nhiên, thay vì tính trực tiếp ma trận Hessian nghịch đảo (H^{-1} , công thức 3.8), Adagrad chỉ thực hiện xấp xỉ tương đối H^{-1} và tính bước cập nhật tiếp theo bằng công thức 3.9:

$$x_{t+1} = \prod_X^{diag(G_t)^{1/2}} (x_t - \eta \cdot diag(G_t)^{-1/2} \cdot g_t) \quad (3.9)$$

với g_t là véc-tơ gradient ở thời điểm t và $G_t = \sum_{i=0}^t g_i \cdot g_i^\top$. Công thức 3.9 chỉ sử dụng gradient bậc nhất để xấp xỉ đường chéo của ma trận Hessian. Vì các bước tính đường chéo và căn bậc hai của ma trận G_t có chi phí

tuyến tính nên thuật toán Adagrad vẫn đảm bảo hiệu quả tính toán.

- Vì gradient g là một véc-tơ, biểu thức $g \cdot g^\top$ sẽ cho kết quả là một ma trận vuông kích thước $M \times M$ với mỗi phần tử là tích của từng cặp hướng trong véc-tơ gradient:

$$G = g \cdot g^\top = \begin{bmatrix} g_1^2 & g_1 \cdot g_2 & \cdots & g_1 \cdot g_M \\ g_2 \cdot g_1 & g_2^2 & \cdots & g_2 \cdot g_M \\ \vdots & \vdots & \ddots & \vdots \\ g_M \cdot g_1 & g_M \cdot g_2 & \cdots & g_M^2 \end{bmatrix} \quad (3.10)$$

- Từ kết quả công thức 3.10, đường chéo của ma trận G là véc-tơ $[g_1^2 \ g_2^2 \ \cdots \ g_M^2]$, hay bình phương của các giá trị đạo hàm riêng tại các hướng. Từ đó, ta có công thức tính G_t 3.11 và bước cập nhật của thuật toán Adagrad được viết lại thành công thức 3.12:

$$G_t = G_{t-1} + g_t^2 \quad (3.11)$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t} + \epsilon} \cdot g \quad (3.12)$$

Tại mỗi bước cập nhật t , công thức 3.11 tính G_t bằng cách cộng dồn bình phương của gradient theo từng hướng. Như vậy, các trọng số được cập nhật nhiều và thường xuyên sẽ có giá trị tương ứng trong G_t lớn, trong khi các trọng số ít được cập nhật sẽ có giá trị nhỏ hơn. Do siêu tham số tỉ lệ học được chia cho căn bậc hai của G_t (công thức 3.12), nên tỉ lệ học của các trọng số có giá trị lớn trong G_t bị tiêu giảm, đồng thời tăng cường tỉ lệ học cho các trọng số có giá trị trong G_t nhỏ. Khả năng điều chỉnh tỉ lệ học tùy theo mức độ và tần suất cập nhật cho mỗi trọng số của Adagrad đã tạo ra nhóm phương pháp “tỉ lệ học thích ứng”. Việc điều chỉnh tỉ lệ học cho từng trọng số giúp mô hình chú ý nhiều hơn đến các đặc trưng

hiếm gặp của dữ liệu.

Tuy nhiên, do g_t^2 luôn luôn không âm nên giá trị của G_t luôn luôn tăng dần, khiến cho tỉ lệ học bị giảm dần. Khi quá trình huấn luyện càng kéo dài, tỉ lệ học sẽ bị suy biến đến mức không thể thực hiện được những bước cập nhật hiệu quả. Để khắc phục vấn đề này, Tijmen Tieleman và Geoffrey Hinton đã đề xuất thuật toán RMSprop trong bài giảng trên Coursera[29]. Thuật toán RMSprop thay đổi công thức 3.11, sử dụng một tỉ lệ suy biến γ cho G_t để ưu tiên các giá trị gradient hiện tại hơn và quên dần các giá trị cũ. Bước tính G_t của RMSprop trở thành công thức 3.13:

$$G_t = \gamma \cdot G_{t-1} + (1 - \gamma) \cdot g_t^2 \quad (3.13)$$

Tỉ lệ suy biến γ ưu tiên các giá trị gradient gần với hiện tại giống như hệ số β trong thuật toán momentum. Tuy nhiên, tỉ lệ suy biến γ khác với β ở chỗ nó không chỉ có tác dụng suy biến mà còn sử dụng như một hệ số tỷ lệ điều chỉnh độ lớn của bước cập nhật. Có nghĩa là nếu gán $\gamma = 0.99$ thì bên cạnh việc giá trị gradient bị suy biến thì tổng bình phương của các gradient sẽ được điều chỉnh với tỷ lệ là $\sqrt{1 - \gamma} = \sqrt{1 - 0.99} = 0.1$. Như vậy, từ công thức 3.13 sẽ cho bước cập nhật lớn hơn gấp 10 lần bước cập nhật của Adagrad với cùng một tỉ lệ học. Nhờ sự thay đổi này mà RMSprop có thể giữ được bước cập nhật đủ lớn khi quá trình huấn luyện kéo dài.

Công thức 3.9 chỉ sử dụng đường chéo trong ma trận G để thực hiện xấp xỉ do đó bước cập nhật phụ thuộc vào tính độc lập của từng trọng số. Vì ta thích ứng tỉ lệ học riêng cho từng trọng số nên tính độc lập giữa các trọng số là cần thiết để đảm bảo sự thay đổi trong tỉ lệ học luôn mang lại hiệu quả tốt. Ngoài ra, ta cũng có thể xem các giá trị bình phương đạo hàm riêng của từng hướng là một hình chiếu của véc-tơ G lên trực trọng số tương ứng. Do đó, độ lớn của véc-tơ hình chiếu sẽ lớn nhất nếu véc-tơ G song song với trực và độ lớn sẽ ngày càng giảm khi góc hợp bởi véc-tơ G và trực trọng số tiến gần đến 90° . Từ những lý do trên mà các phương

pháp tỉ lệ học thích ứng hoạt động tốt trong các trường hợp hướng nhạy cảm là hướng song song với trục trọng số hay nói cách khác, các đặc trưng của dữ liệu độc lập tuyến tính với nhau.

3.3 Thuật toán Adam

Momentum tăng cường hướng tối ưu bằng quán tính còn RMSprop hạn chế di chuyển về hướng dao động thông qua điều chỉnh tỉ lệ học cho từng trọng số tương ứng. Lấy ý tưởng từ hai thuật toán trên, Diederik P. Kingma và Jimmy Lei Ba đề xuất thuật toán Adam vừa sử dụng quán tính để tăng tốc và giảm dao động vừa điều chỉnh tỉ lệ học tương ứng với từng trọng số[14].

$$\begin{aligned} m_t &= \beta_1 \cdot m_t + (1 - \beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_t + (1 - \beta_2) \cdot g_t^2 \end{aligned} \tag{3.14}$$

Thuật toán Adam cũng thực hiện xấp xỉ ma trận Hessian như các thuật toán Adagrad, RMSprop và kết hợp một lượng quán tính, tương tự với thuật toán Momentum. Tuy nhiên, khác với Momentum, Adam không tích luỹ quán tính phụ thuộc vào độ lớn của bước cập nhật mà thực hiện xấp xỉ dựa vào các giá trị “moment” của gradient. Moment bậc k của một biến ngẫu nhiên X là giá trị kỳ vọng $\mathbf{E}[X^k]$ với $k \in N$. Từ đó, ta có moment bậc nhất, hay trung bình, và moment bậc hai, hay phương sai, của gradient g_t lần lượt là $\mathbf{E}[g_t]$, $\mathbf{E}[g_t^2]$. Hai giá trị này lần lượt được thuật toán Adam xấp xỉ bằng cách tích luỹ trung bình chạy của gradient thông qua véc-tơ m_t và trung bình chạy của bình phương gradient với véc-tơ v_t với hệ số decay $\beta_1, \beta_2 \in [0, 1)$ (công thức 3.14). Trong thực tế huấn luyện ta thường khởi tạo các véc-tơ m_t, v_t bằng giá trị 0 nên các giá trị xấp xỉ thường “bias” về giá trị khởi tạo trong các bước cập nhật đầu, đặc biệt khi β_1 và $\beta_2 \approx 1$.

Ta có thể tính véc-tơ v_t tại mỗi bước cập nhật t dựa vào công thức 3.15:

$$v_t = (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} g_i^2 \quad (3.15)$$

Cũng như các thuật toán sử dụng gradient của minibatch khác, thuật toán Adam sử dụng gradient có độ sai khác so với gradient thật. Để xấp xỉ moment bậc hai của gradient gần với moment bậc hai $\mathbf{E}[g_t^2]$ của gradient thật nhất có thể, ta thực hiện bước “bias-correction” giúp thuật toán thực hiện xấp xỉ tốt hơn tại các bước cập nhật đầu tiên.

$$\mathbf{E}[v_t] = \mathbf{E}[(1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} g_i^2] = \mathbf{E}[g_t^2] \cdot (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} + \zeta \quad (3.16)$$

Trong công thức 3.16, độ lỗi ζ đến từ nhiều khi chọn tập con một cách ngẫu nhiên và có thể khắc phục bằng cách chọn siêu tham số β_2 sao cho giá trị g_t^2 ở các bước cập nhật quá xa trong quá khứ có độ ưu tiên nhỏ hơn giá trị g_t^2 ở bước cập nhật gần nhất. Thông thường, giá trị β_2 được chọn rất gần với 1 và được gán mặc định là $\beta_2 = 0.999$. Công thức 3.16 sẽ không mất đi tính tổng quát nếu ta thay thế tổng $\sum_{i=1}^t \beta_2^{t-i} = \beta^t \sum_{i=1}^t (\beta_2^{-1})^i$. Áp dụng công thức $\sum_{i=1}^t r^i = \frac{r(1-r^t)}{1-r}$ ta được một biểu thức mới $\beta_2^t (1 - \beta_2) \frac{\beta_2^{-1}(1-\beta_2^{-1,t})}{1-\beta_2^{-1}}$ với $r = \beta_2^{-1}$. Rút gọn biểu thức trên ta có $(1 - \beta_2) \frac{\beta_2^t(1-\beta_2^{-t})}{\beta_2(1-\beta_2^{-1})} = (1 - \beta_2) \frac{(\beta_2^t-1)}{(\beta_2-1)} = 1 - \beta_2^t$ và xấp xỉ của $\mathbf{E}[v_t]$ sẽ bằng với $\mathbf{E}[g_t^2] \cdot (1 - \beta_2^t) + \zeta$. Vậy để thực hiện bias-correction, ta lấy véc-tơ v_t chia cho giá trị $(1 - \beta_2^t)$ để khắc phục hiện tượng giá trị xấp xỉ bias về giá trị khởi tạo. Tương tự như vậy ta cũng thực hiện bước bias-correction với véc-tơ m_t bằng cách lấy m_t chia cho $(1 - \beta_1^t)$. Sau bước bias-correction ta được \hat{m}_t , \hat{v}_t như công thức 3.17.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \quad (3.17)$$

Đối với các thuật toán sử dụng trung bình chạy thì việc thực hiện bước bias-correction là cần thiết cho xấp xỉ tại những bước đầu tiên tốt hơn. Vì trong các bước đầu tiên, tương đương với t nhỏ, trung bình chạy không đủ giá trị để cho ra một xấp xỉ chính xác mà thường bias về giá trị khởi tạo. Giai đoạn này được gọi là giai đoạn “warm-up”. Việc thực hiện bias-correction sẽ rút ngắn thời gian warm-up và giúp thuật toán hội tụ nhanh hơn. Hơn nữa, khi xử lý các tập dữ liệu thưa, việc lựa chọn hệ số β_2 gần với 1 là điều cần thiết vì khi đó giá trị của gradient là rất nhỏ tại nhiều bước cập nhật liên tiếp nên các giá trị trong quá khứ vẫn còn nhiều thông tin cần được tích luỹ. Tuy nhiên, việc lựa chọn giá trị β_2 khiến cho xấp xỉ tại những bước đầu của trung bình chạy rất tệ và thiếu bước bias-correction sẽ cho các bước nhảy đầu rất lớn gây nguy cơ phân kỳ. Đây là một trong những điểm quan trọng giúp Adam hiệu quả hơn các thuật toán RMSprop hay Adagrad.

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (3.18)$$

Công thức 3.18 cho ta bước cập nhật của thuật toán Adam với độ dài bước nhảy $\Delta t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$. Trong đó ϵ đảm bảo phân số $\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ có nghĩa ($\sqrt{\hat{v}_t} + \epsilon \neq 0$ (không bị chia cho 0) nên sẽ không mất đi tính tổng quát khi viết $\Delta t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$ với $\hat{v}_t > 0$). Vì \hat{m}_t và \hat{v}_t đều là các trung bình chạy lần lượt được xác định bởi hệ số β_1 và β_2 nên $\hat{m}_t \leq (1-\beta_1)$ và $\hat{v}_t \leq (1-\beta_2)$. Từ đó, ta có giới hạn trên của $\Delta t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \leq \alpha \frac{1-\beta_1}{\sqrt{1-\beta_2}}$ với $(1-\beta_1) > \sqrt{1-\beta_2}$. Trong thực tế huấn luyện, các giá trị β_1 và β_2 được chọn thường không thoả điều kiện trên, như giá trị mặc định cho ta $1-\beta_1 = 1-0.9 = 0.1 = \sqrt{1-\beta_2} = \sqrt{1-0.99}$. Do đó, ta có giới hạn trên của $\Delta t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \leq \alpha \frac{1-\beta_1}{\sqrt{1-\beta_2}} \leq \alpha$ do $\frac{1-\beta_1}{\sqrt{1-\beta_2}} \leq 1$. Vì lý do đó mà tỉ lệ học α được xem như một khoảng tin cậy xung quanh điểm dữ liệu đang xét đảm bảo rằng xấp xỉ gradient vẫn còn đủ thông tin để xác định hướng cho điểm mới có độ lỗi nhỏ hơn điểm đang xét.

Tính chất này giúp Adam giảm thiểu số lượng siêu tham số phải tính

chỉnh trước khi huấn luyện mô hình. Trong đa số các bài toán máy học phức tạp, việc đi tìm một tỉ lệ học phù hợp là không dễ dàng do còn phụ thuộc vào nhiều yếu tố như: kiến trúc mạng, phân bố dữ liệu, cấu trúc dữ liệu,... Tuy nhiên, với đa số bài toán, ta có thể xác định vùng trên bề mặt lỗi mà xác suất chứa cực tiểu là lớn như khởi tạo giá trị trọng số bằng quá trình học không giám sát. Nhờ đó giá trị α có thể được xác định để đến được cực tiểu từ θ_0 trong hữu hạn vòng lặp. Với cách tiếp cận này, thời gian tinh chỉnh cho tham số α sẽ được rút ngắn và một giá trị của α có thể hoạt động tốt trong nhiều bài toán khác nhau, giúp cho việc thiết kế và thử nghiệm mô hình dễ dàng hơn.

Trong bài báo gốc của Adam[14], Diederik P. Kingma và Jimmy Lei Ba cho biết thuật toán Adam là một xấp xỉ của thuật toán “Natural Gradient Descent” (thuật toán Gradient Descent nhưng sử dụng ma trận thông tin Fisher để đo sự sai khác giữa hai phân phối xác suất của tại hai bước cập nhật) và $E[v_t]$ là một xấp xỉ của ma trận thông tin Fisher. Tuy nhiên, có một vài điểm gây bất hợp lý vì ma trận Fisher được xấp xỉ bằng $E[v_t]^{-1}$ trong khi Adam lại sử dụng $E[v_t]^{-1/2}$. Việc lý giải tại sao lại sử dụng số mũ $-\frac{1}{2}$ thay cho -1 cũng chưa được giải thích rõ ràng. Mặc dù vậy, đã có một số thực nghiệm chứng minh rằng việc sử dụng số mũ là -1 gây ra bất ổn định và cho ra kết quả tệ hơn. Sebastian Ruder cho rằng việc sử dụng mũ $-\frac{1}{2}$ cho hiệu quả tốt hơn với thuật toán Adagrad[21]; Staib và cộng sự cũng có một thí nghiệm chứng minh tính bất ổn định của thuật toán RMSProp khi sử dụng số mũ là -1 thay cho $-\frac{1}{2}$ [26]. Trong cả hai trường hợp số mũ kể trên, các phương pháp tỉ lệ học thích ứng nói chung và Adam nói riêng không làm thay đổi hướng cập nhật của gradient chỉ có kích thước bước cập nhật được thay đổi bằng thông tin về độ cong mặt lỗi[26].

Chương 4

Các kết quả thí nghiệm

Trong chương này, chúng tôi trình bày các kết quả thí nghiệm nhằm đánh giá những nội dung đã trình bày ở Chương 3. Cho các thí nghiệm về nguyên lý, chúng tôi sử dụng dữ liệu được tạo ngẫu nhiên và hàm lỗi Mean Squared Error; với các thí nghiệm thực tế, chúng tôi sử dụng bộ dữ liệu MNIST và CIFAR10 và hàm lỗi Cross Entropy. Kết quả thí nghiệm cho thấy thuật toán mà chúng tôi cài đặt có thể xấp xỉ kết quả mà bài báo công bố, tuy nhiên chúng tôi nhận thấy rằng bộ siêu tham số được sử dụng để tái tạo kết quả có thể không cho kết quả tốt nhất cho tất cả thuật toán. Ngoài ra, các kết quả thí nghiệm cũng cho thấy các trường hợp mà các thuật toán khác gặp khó khăn, và cách mà Adam vượt qua các khó khăn đó. Cuối cùng, các thí nghiệm cho thấy tốc độ tối ưu hóa của các thuật toán trên các mô hình mạng nơ-ron thực tế với nhiều tầng ẩn gồm các cấu trúc khác nhau.

4.1 Các thiết lập thí nghiệm

Chúng tôi thực hiện hai loại thí nghiệm: thí nghiệm nguyên lý và thí nghiệm thực tế. Trong các thí nghiệm nguyên lý, chúng tôi sử dụng hàm số để tạo ra dạng địa hình bề mặt lỗi mong muốn. Ngoài ra, chúng tôi cũng tạo một bộ dữ liệu ngẫu nhiên cho các thí nghiệm sử dụng minibatch. Với

các thí nghiệm thực tế, chúng tôi sử dụng nhiều bộ dữ liệu khác nhau cùng với các kiến trúc mạng nơ-ron nhiều tầng ẩn được sử dụng để giải quyết các bài toán ứng dụng hiện nay. Những bộ dữ liệu thực tế mà chúng tôi sử dụng:

- MNIST [16]: Bộ dữ liệu này gồm các ảnh xám của các chữ số viết tay từ 0 đến 9 có kích thước 28x28. Tập dữ liệu gồm 60000 ảnh huấn luyện và 10000 ảnh kiểm tra.
- CIFAR10 [15]: Bộ dữ liệu này gồm các ảnh màu của 10 lớp đối tượng có kích thước 32x32. Tập dữ liệu có tổng cộng 60000 ảnh, mỗi lớp đối tượng có 6000 ảnh. Trong đó 50000 ảnh được chọn làm ảnh huấn luyện và 10000 ảnh kiểm tra. Bộ dữ liệu này và MNIST là những bộ dữ liệu cơ bản phổ biến để huấn luyện các thuật toán học máy và mạng nơ-ron nhiều tầng ẩn đơn giản.
- ImageNet [9]: Đây là một trong những bộ dữ liệu hình ảnh lớn nhất hiện nay, thường được dùng để kiểm tra khả năng của các phương pháp thị giác máy tính mới nhất. Phiên bản của bộ dữ liệu mà chúng tôi sử dụng là phiên bản được sử dụng trong thử thách ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012), với hơn 1.2 triệu ảnh được gán nhãn vào 1000 lớp đối tượng được sử dụng để huấn luyện, và 50000 ảnh để kiểm thử.
- Penn Tree Bank [18]: Đây là bộ dữ liệu do Marcus và cộng sự tổng hợp gồm 929 000 từ dùng để huấn luyện, 73 000 từ cho tập kiểm thử và 82 000 từ để kiểm tra. Tập dữ liệu gồm có 10 000 từ trong tập từ điển. Tập dữ liệu đã được qua xử lý được lấy từ trang web của Tomas Mikolov (<http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz>).

Chúng tôi sử dụng ngôn ngữ lập trình Python và thư viện Numpy cho các thí nghiệm nguyên lý, thư viện Pytorch cho các thí nghiệm thực tế.

Tên thuật toán	Tỉ lệ học	Momentum / $Beta_1$	Alpha / $Beta_2$
SGD	[0.0001, 0.1]	-	-
Momentum	[0.0001, 0.1]	[0.1, 0.99]	-
Adagrad	[0.0001, 0.1]	-	-
RMSprop	[0.0001, 0.1]	-	[0.9, 0.999]
Adam	[0.0001, 0.1]	[0.1, 0.99]	[0.9, 0.999]

Bảng 4.1: Khoảng giá trị để dò tìm siêu tham số tốt nhất của các thuật toán tối ưu.

Cả thư viện Numpy và Pytorch đều cung cấp khả năng tăng tốc thực thi bằng việc véc-tơ hóa tính toán trên nền C/C++. Thư viện Numpy tập trung vào các chức năng tính toán đại số trên CPU, phù hợp với các thí nghiệm đơn giản; trong khi thư viện Pytorch là một thư viện máy học có thể xây dựng các mạng nơ-ron nhiều tầng ẩn phức tạp cùng với khả năng thực thi song song trên GPU. Loại GPU mà chúng tôi sử dụng là NVIDIA RTX 2080, ngoài ra chúng tôi cũng sử dụng NVIDIA Tesla T4 và Cloud Tensor Processing Unit trên nền tảng Google Colaboratory.

Trong tất cả các thí nghiệm, tất cả các thuật toán tối ưu đều có chung điểm xuất phát: bộ trọng số ban đầu đầu của mạng nơ-ron nhiều tầng ẩn sau khi khởi tạo ngẫu nhiên lần đầu sẽ được lưu lại; và với mỗi thuật toán tối ưu được thử nghiệm, bộ trọng số này sẽ được nạp lại vào mạng nơ-ron rồi từ đó mới bắt đầu tiến hành tối ưu và ghi nhận lại kết quả. Điều này giúp đảm bảo sự công bằng cho tất cả các thuật toán khi tiến hành thử nghiệm, tránh được trường hợp một thuật toán nào đó "may mắn" được bắt đầu ở một điểm thuận lợi hơn và nhờ vậy mà hội tụ nhanh hơn.

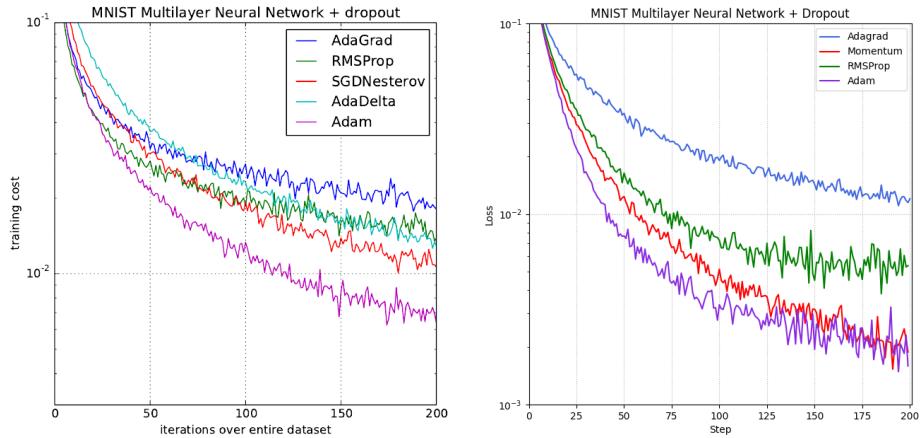
4.2 Các kết quả thí nghiệm

4.2.1 Kết quả của thuật toán cài đặt so với bài báo

Trong phần này, chúng tôi trình bày kết quả của các thuật toán tối ưu mà chúng tôi cài đặt. Cụ thể, chúng tôi so sánh kết quả của các thuật toán do chúng tôi cài đặt lại trên nền thư viện Pytorch với kết quả được công bố trong bài báo trong thí nghiệm Multi-layer Neural Network trên tập dữ liệu MNIST và thí nghiệm Convolutional Neural Network trên tập dữ liệu CIFAR10. Việc tái tạo các kết quả được công bố gấp nhiều khó khăn do bản chất ngẫu nhiên của SGD và dropout, cộng với việc tác giả không công bố các siêu tham số cho các thí nghiệm cũng như các cài đặt liên quan khác. Bài báo cũng không đề cập các khoảng tìm kiếm cho quá trình tìm siêu tham số tối ưu nhất của mỗi thuật toán, vì vậy chúng tôi đã tham khảo các công trình nghiên cứu khác sử dụng các tối ưu này, và tổng hợp các giá trị nhỏ nhất và lớn nhất cho từng siêu tham số của mỗi thuật toán để làm khoảng tìm kiếm (bảng 4.1). Bài báo thực hiện thí nghiệm với thuật toán Nesterov momentum, Adagrad, RMSprop, AdaDelta và Adam; tuy nhiên chúng tôi nhận thấy rằng Nesterov momentum và AdaDelta không liên quan trực tiếp đến thuật toán Adam mà khóa luận tập trung tìm hiểu, vì vậy chúng tôi sẽ tập trung thực hiện thí nghiệm trên các thuật toán SGD, Momentum, Adagrad, RMSprop, và Adam.

Trong thí nghiệm “Multi-layer Neural Network”, kết quả mà chúng tôi cài đặt so với kết quả được công bố trong bài báo xấp xỉ nhau về đường đi và vị trí tương đối giữa các thuật toán với nhau (hình 4.1). Tuy nhiên, giá trị loss trong cài đặt của chúng tôi cũng thấp hơn đáng kể so với kết quả trong bài báo.

Nhìn chung, thuật toán Adam có tốc độ tối ưu độ lỗi nhanh nhất trong tất cả các thuật toán được thử nghiệm. Điểm yếu của Adagrad về tỉ lệ học luôn giảm dần được thể hiện rõ khi Adagrad chuyển hướng đi ngang dần mặc dù độ lỗi vẫn cao trong khi các thuật toán khác tiếp tục đi xuống.



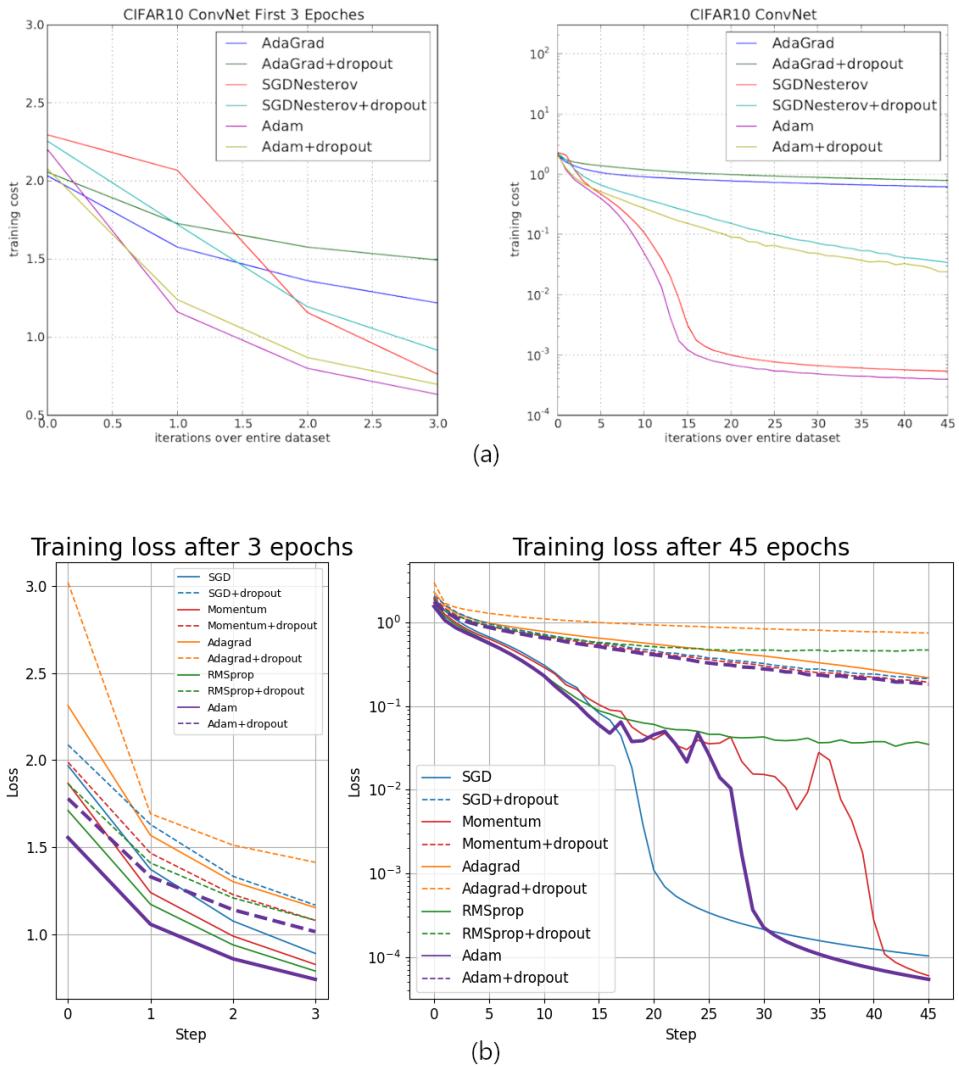
Hình 4.1: Kết quả thí nghiệm Multi-layer Neural Network giữa thuật toán mà chúng tôi cài đặt so với bài báo. Bên trái: kết quả mà bài báo công bố. Bên phải: kết quả do chúng tôi tự cài đặt lại.

Điều đó cho thấy rằng mô hình vẫn đang trong quá trình học dữ liệu, nhưng vì tỉ lệ học quá nhỏ nên ở mỗi bước, lượng cập nhật trọng số không đủ để cải thiện mô hình. Vấn đề tỉ lệ học giảm dần do cộng dồn gradient bình phương của Adagrad được RMSprop khắc phục bằng một tỉ lệ suy biến, giúp RMSprop bắt kịp gần hơn với Momentum và Adam. Thuật toán Momentum là thuật toán có kết quả khác biệt nhất giữa cài đặt của chúng tôi và cài đặt trong bài báo. Nếu như trong bài báo, Momentum khá xấp xỉ với RMSprop thì chúng tôi lại nhận được rằng Momentum luôn luôn có độ lỗi tốt hơn RMSprop, và thậm chí còn bắt kịp với Adam ở những bước cuối cùng.

Trong thí nghiệm “Convolutional Neural Network”, chúng tôi cũng thực hiện tìm bộ siêu tham số đem lại kết quả cuối cùng thấp nhất cho mỗi thuật toán. Tuy nhiên, sau khi đã tìm được những bộ siêu tham số cho tất cả các thuật toán và thực hiện thí nghiệm, chúng tôi thấy được rằng hầu hết các thuật toán trong cài đặt của chúng tôi đều giúp mô hình đạt được độ lỗi thấp hơn đáng kể so với bài báo khi không dùng dropout. Các giá trị siêu tham số được sử dụng được trình bày chi tiết trong bảng A.2.

Từ hình 4.2, chúng ta có thể thấy rằng bề mặt lỗi của thí nghiệm này có

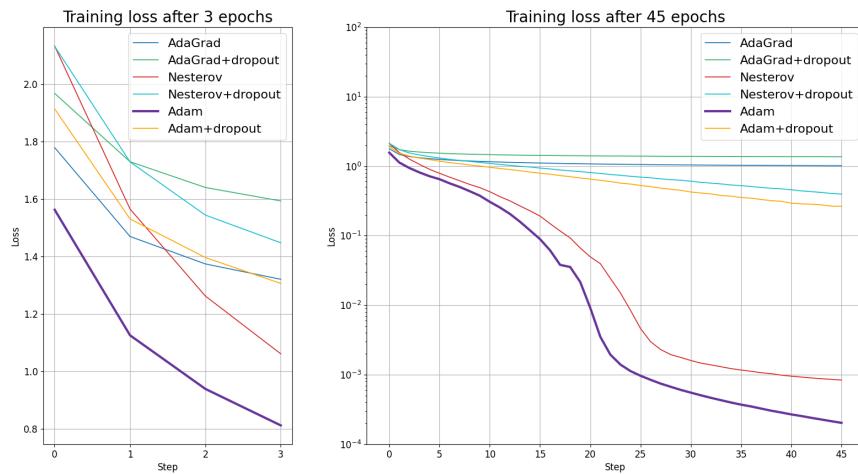
một đoạn dốc xuống khá rõ rệt, và các thuật toán SGD, Momentum cũng như Adam đều giảm độ lỗi rất nhanh khi tìm thấy đoạn dốc ấy. Trong bài báo gốc, một số thuật toán tìm thấy đoạn dốc này ngay từ những epoch đầu tiên, trong khi cài đặt của chúng tôi chỉ bắt đầu giảm mạnh từ khoảng epoch thứ 15. Trong kết quả của chúng tôi, SGD là thuật toán đầu tiên tìm ra đoạn dốc, nhưng độ lỗi tại epoch cuối lại hơi cao hơn so với Momentum và Adam, có thể là do SGD bị giới hạn bởi tỉ lệ học trong khi hai thuật toán kia có cơ chế tăng tốc và đạt được độ lỗi thấp.



Hình 4.2: Kết quả thí nghiệm Convolutional Neural Network giữa thuật toán mà chúng tôi cài đặt so với bài báo. (a): kết quả mà bài báo công bố. (b): kết quả do chúng tôi tự cài đặt lại.

Ngoài việc tìm bộ siêu tham số cho kết quả tốt nhất, chúng tôi cũng cố gắng tái tạo kết quả mà bài báo đã công bố. Chúng tôi thực hiện quá trình tái tạo này để kiểm tra tính khả thi của các kết quả mà bài báo đã công bố. Hình 4.3 là kết quả sát nhất với bài báo mà chúng tôi có thể tái tạo được từ khoảng tìm kiếm mà chúng tôi sử dụng. Có thể thấy được rằng, đường đi của các thuật toán trong hình 4.2a và 4.3 có dạng khá giống

nhau. Mặc dù có thể tái tạo được hình dạng biểu đồ của các thuật toán, độ lỗi cuối cùng của các thuật toán trong hình 4.3 vẫn cao hơn so với kết quả tốt nhất mà chúng tôi ghi nhận được ở hình 4.2b. Kết quả độ lỗi và thời gian thực thi được trình bày trong bảng 4.2.



Hình 4.3: Kết quả thí nghiệm Convolutional Neural Network giữa thuật toán mà chúng tôi cài đặt với bộ siêu tham số mô phỏng kết quả của bài báo.

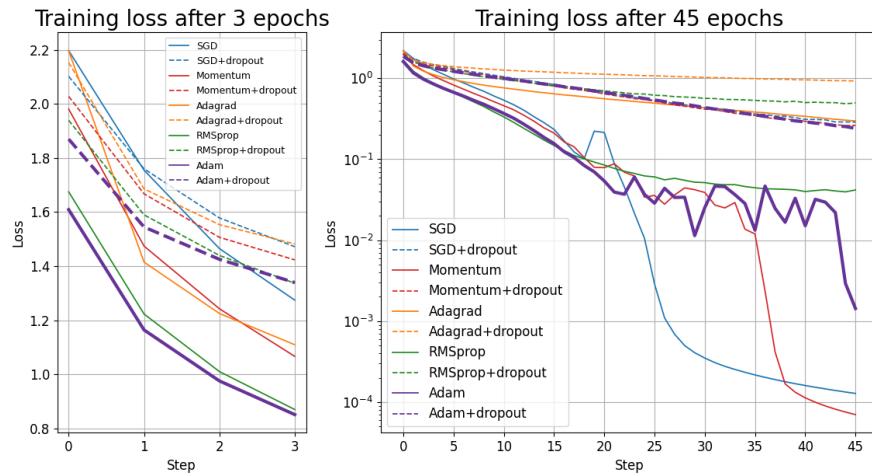
Ngoài lí do tác giả không công bố siêu tham số, cũng như tính chất ngẫu nhiên của SGD, một nguyên nhân khả thi khác cho sự khác biệt giữa kết quả mà chúng tôi ghi nhận được so với kết quả mà bài báo công bố là bản chất cài đặt của thư viện. Mỗi thư viện có cách xử lý tính toán riêng dẫn đến kết quả có phần lệch nhau. Vì tác giả cũng không nói rõ thư viện được sử dụng trong bài báo, nên thư viện Pytorch mà chúng tôi sử dụng có thể sẽ ra kết quả khác so với bài báo.

Trong bài báo Adam, tác giả sử dụng kĩ thuật làm trắng ảnh (“whitening”) để tiền xử lý cho ảnh đầu vào. Làm trắng ảnh là quá trình tách biệt các mối liên hệ giữa các đặc trưng của ảnh, khiến các đặc trưng này trở thành độc lập [13]. Các đặc trưng độc lập và trùng với trực của các trọng số là một điều kiện có lợi cho các thuật toán tỉ lệ học thích ứng. Để kiểm chứng lợi thế này, chúng tôi lặp lại thí nghiệm trên, nhưng bỏ qua công

Tên thuật toán	Thời gian thực hiện (giây)	Độ lỗi thấp nhất	Độ lỗi thấp nhất (có làm trắng ảnh)
SGD	6.02 ±0.07	0.0001	0.0001
SGD+dropout	6.04 ±0.07	0.2875	0.2139
Momentum	6.10 ± 0.07	0.00007	0.00006
Momentum+dropout	6.08 ± 0.07	0.2604	0.1914
Adagrad	6.14 ± 0.09	0.2973	0.2178
Adagrad+dropout	6.16 ± 0.09	0.9237	0.7496
RMSprop	6.16 ± 0.07	0.0395	0.03307
RMSprop+dropout	6.19 ± 0.07	0.4855	0.4512
Adam	6.19±0.07	0.0014	0.000054
Adam+dropout	6.23±0.07	0.2403	0.1824

Bảng 4.2: Kết quả và thời gian thực hiện một epoch của các thuật toán.

đoạn làm trắng ảnh.



Hình 4.4: Kết quả thí nghiệm Convolutional Neural Network mà không thực hiện làm trắng ảnh.

Kết quả của hình 4.4 cho thấy độ lỗi Adam cao hơn khá nhiều khi bỏ qua bước làm trắng ảnh trong khi các thuật toán khác không bị ảnh hưởng nhiều. Như vậy, chúng ta có thể thấy việc làm trắng ảnh đầu vào sẽ giúp ích đáng kể cho quá trình huấn luyện khi sử dụng thuật toán Adam.

Về mặt thời gian tính toán, bảng 4.2 cho thấy với cùng một kích thước minibatch, có thể dễ hiểu khi SGD cho thời gian tính toán nhanh nhất vì thuật toán này chỉ tính gradient và cập nhật trọng số. Với Momentum, bước tính véc-tơ quán tính v_t trước khi cập nhật trọng số làm tăng thời gian thực hiện mỗi epoch thêm khoảng 15%. Bước tính đường chéo ma trận G_t của các thuật toán tỉ lệ học thích ứng tồn tại nhiều thời gian hơn đáng kể, vì trong bước này còn có thao tác bình phương các phần tử trong véc-tơ gradient trước khi cộng dồn (thuật toán RMSprop có thêm bước nhân các phần tử này với hệ số suy biến). Adam kết hợp các bước tính quán tính trong Momentum và thích ứng tỉ lệ học, vì vậy Adam thời gian thực thi chậm nhất trong tất cả các thuật toán.

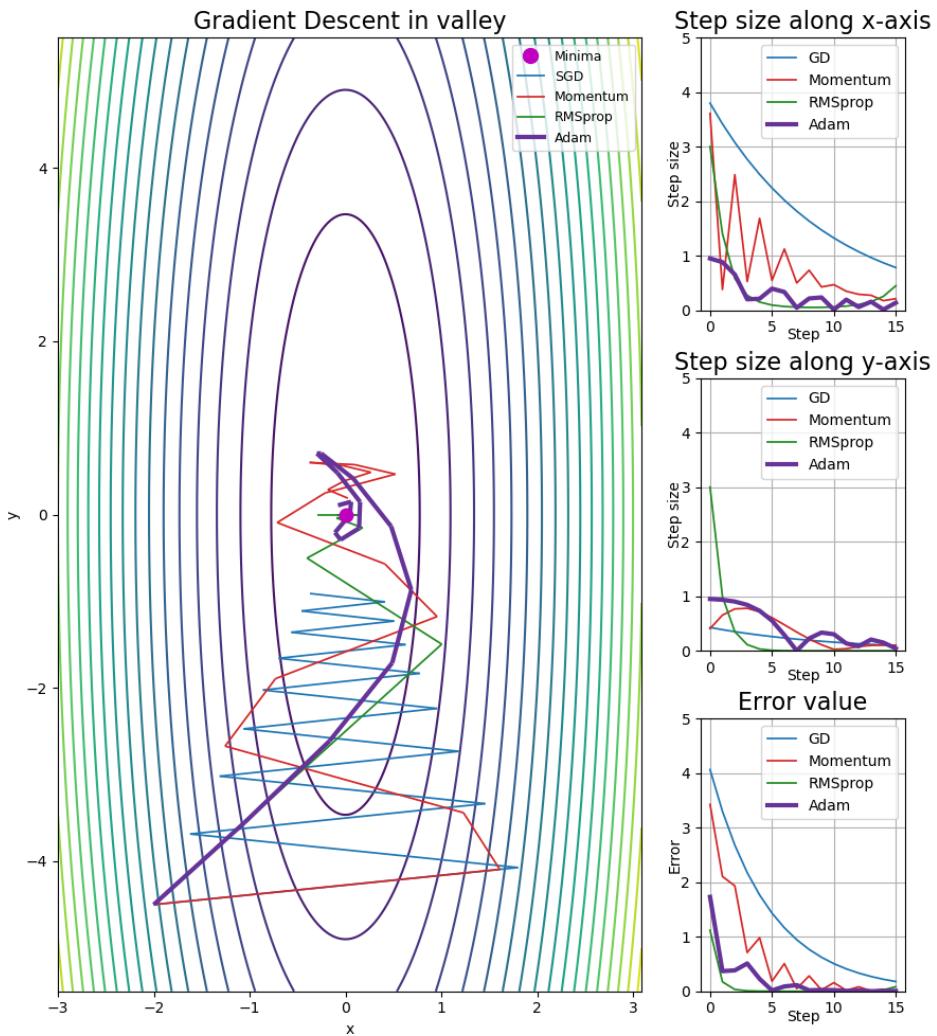
Nhìn chung, chúng tôi có thể tái tạo các kết quả thí nghiệm mà bài báo công bố. Tuy nhiên, các kết quả tốt nhất của chúng tôi có độ lỗi thấp hơn đáng kể so với bài báo, đặc biệt là trong thí nghiệm Convolutional Neural Network. Vì chúng tôi không biết chính xác quá trình cài đặt và các siêu tham số đã được sử dụng trong bài báo gốc nên chúng tôi không thể khẳng định nguyên nhân của sự chênh lệch này. Chúng tôi cũng mở rộng thí nghiệm Convolutional Neural Network để tìm hiểu ảnh hưởng của việc làm tách biệt các đặc trưng của dữ liệu thông qua thao tác làm trắng ảnh lên quá trình huấn luyện.

4.2.2 Phân tích độ lớn bước cập nhật của các trọng số trong rãnh hẹp

Trong thí nghiệm này, chúng tôi kiểm chứng các khó khăn của dạng địa hình rãnh hẹp và cách mà các thuật toán khác nhau di chuyển trong đó. Vùng rãnh hẹp là vùng mà độ cong giữa các chiều có sự chênh lệch rất lớn. Cụ thể hơn, một số hướng sẽ có độ cong rất lớn trong khi các hướng khác lại rất bằng phẳng. Vì vậy, để có thể di chuyển hiệu quả trong địa hình này, chúng ta cần giảm kích thước bước cập nhật trên các chiều có độ cong lớn để tránh hiện tượng dao động, đồng thời cập nhật những bước

dài hơn trên các chiều bằng phẳng có giá trị gradient nhỏ.

Chúng tôi sử dụng một hàm lỗi giả lập có 2 tham số với sự chênh lệch độ cong trên mỗi trục là rất lớn để tạo thành rãnh hẹp. Gradient mà các thuật toán sử dụng cũng sẽ là gradient của hàm mục tiêu tại vị trí đang xét. Vì vậy, đây có thể được coi như gradient của cả tập dữ liệu. Tất cả các thuật toán đều có cùng một điểm xuất phát.



Hình 4.5: Đường đi, độ lớn bước cập nhật theo từng chiều, và độ lỗi của các thuật toán trong quá trình tối ưu hàm giả lập.

Hình 4.5 cho thấy cách mỗi thuật toán di chuyển trong rãnh hẹp. GD rõ ràng là thuật toán gặp nhiều khó khăn nhất khi liên tục bị dao động giữa 2 bên vách. Độ lớn bước cập nhật trên trục x của GD rất lớn, trong

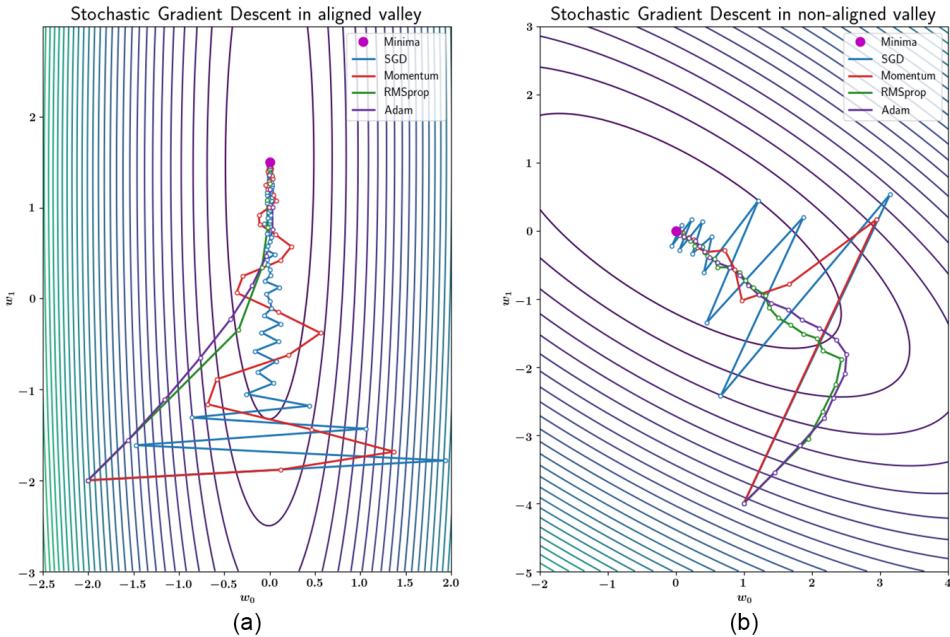
khi lượng cập nhật trên trục y lại quá nhỏ khiến GD không đến được điểm cực tiểu. Momentum ban đầu cũng gặp khó khăn tương tự, nhưng nhờ quán tính kìm hãm lượng dao động trên trục x đồng thời đẩy nhanh hơn trên y nên Momentum di chuyển được tốt hơn. Với các thuật toán tỉ lệ học thích ứng, chúng ta có thể thấy RMSprop và Adam tối ưu độ lớn bước cập nhật trên cả trục x và y để di chuyển tốt hơn theo chiều dài của rãnh hẹp. Các biểu đồ độ lớn bước cập nhật cũng cho thấy rõ cách Adam kết hợp Momentum và RMSprop: vừa bước dài hơn theo trục y và hạn chế di chuyển trên trục x, vừa kìm hãm dao động khi hướng của gradient thay đổi. Khi bị lốp qua khỏi điểm cực tiểu, Adam cũng thực hiện đi ngược lại hiệu quả hơn Momentum.

4.2.3 Phân tích đường đi trong rãnh hẹp có hướng trùng và không trùng với trọng số

Từ nội dung đã trình bày ở Chương 3, các thuật toán tỉ lệ học thích ứng thay đổi tỉ lệ học theo từng trọng số riêng biệt. Vì vậy, dạng địa hình rãnh hẹp có các hướng trùng với trục của trọng số sẽ là trường hợp mà các thuật toán tỉ lệ học thích ứng phát huy hiệu quả cao nhất. Ngược lại, trường hợp rãnh hẹp chéo góc so với các trục của trọng số sẽ gây ra nhiều khó khăn nhất cho các thuật toán ấy.

Trong thí nghiệm này, thay vì tối ưu trực tiếp trên hàm giả lập, chúng tôi tạo một bộ dữ liệu ngẫu nhiên gồm 10000 phần tử có phân phối đều trên một trục, với giá trị của các điểm dữ liệu trên trục còn lại được quyết định bởi một hàm có 2 tham số. Mức độ chênh lệch giữa 2 tham số sẽ tạo ra địa hình rãnh hẹp, và sự phụ thuộc giữa các tham số trong hàm sẽ khiến cho các chiều của rãnh hẹp không còn trùng với tham số. Chúng tôi thực hiện “fit” một đường thẳng hồi quy trên bộ dữ liệu này bằng phương pháp Stochastic Gradient Descent. Kích thước minibatch mà chúng tôi sử dụng là 50.

Hình 4.6a cho thấy kết quả tương tự thí nghiệm trước, với Adam và



Hình 4.6: Đường đi của các thuật toán trong rãnh hẹp có hướng trùng với trọng số (a) và không trùng với trọng số (b).

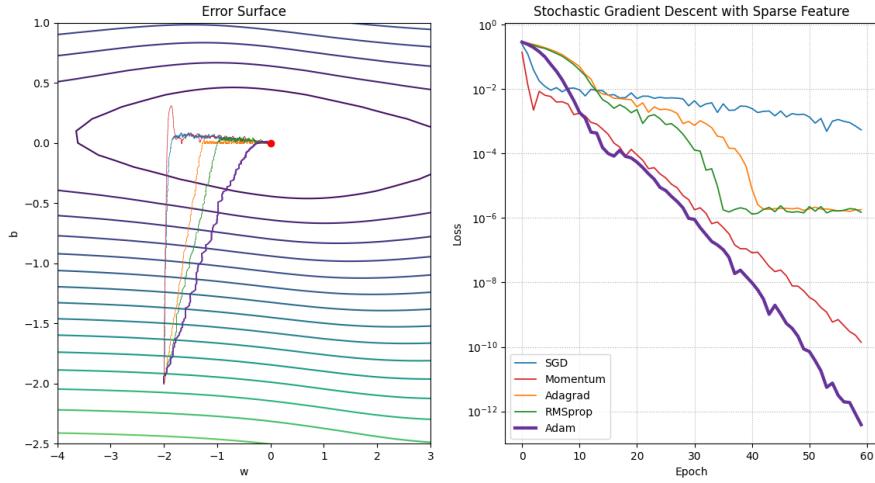
RMSprop bước những bước dài hơn trên hướng bằng phẳng của rãnh hẹp (tương ứng với trục w_1) trong khi SGD và Momentum bị dao động và rất khó di chuyển theo trục w_1 . Tuy nhiên, vì sử dụng gradient của các minibatch để di chuyển nên sự dao động của SGD và Momentum cũng có phần được cải thiện hơn so với thí nghiệm trước sử dụng gradient của bể mặt lỗi.

Trong trường hợp các hướng của rãnh hẹp không trùng với trục của trọng số (hình 4.6b), chúng ta có thể thấy Adam và RMSprop gặp nhiều khó khăn hơn trong việc điều chỉnh tỉ lệ học. Adam và RMSprop chỉ thực hiện những bước cập nhật rất nhỏ. SGD và Momentum hầu như không bị ảnh hưởng bởi hướng của rãnh hẹp.

4.2.4 Vấn đề đặc trưng thừa và nhiễu

Đặc trưng thừa là những đặc trưng mang giá trị 0 hoặc gần bằng 0 tại đa số các điểm dữ liệu; ngược lại, ta có đặc trưng đặc. Số lần các đặc trưng thừa được cập nhật là rất ít hoặc cập nhật với lượng rất nhỏ, từ đó

kéo dài quá trình huấn luyện mạng nơ-ron nhiều tầng ẩn. Thêm nữa, đặc trưng thừa xuất hiện ở hầu hết các mạng nơ-ron sâu hiện nay. Đó là lý do tại sao vấn đề về đặc trưng thừa này cần được giải quyết.



Hình 4.7: Đường đi của các thuật toán trong bề mặt lỗi (trái) và độ lỗi (phải) của từng thuật toán trong trường hợp đặc trưng thừa.

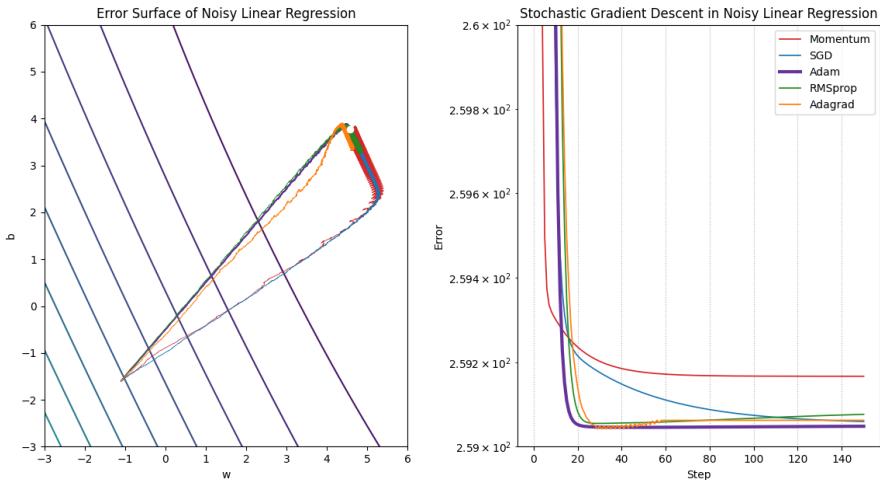
Để quan sát cách các thuật toán tối ưu hoạt động với dữ liệu thừa, thí nghiệm sử dụng dữ liệu là n cặp (x, y) được khởi tạo ngẫu nhiên. Sau đó, chọn 80% trong số n cặp và gán giá trị $x = 0$ với mục đích là khiến cho đặc trưng x thừa. Sử dụng mô hình hồi quy tuyến tính với hàm mục tiêu có dạng $y = wx + b$, các thuật toán được sử dụng để tối ưu hàm chi phí $MSE = \frac{1}{n} \sum_{i=1}^n (y - wx - b)^2$. Kết quả sau 60 epoch được thể hiện như trong hình 4.7b và hình 4.7a thể hiện đường đi của các thuật toán trên bề mặt lỗi.

Các thuật toán tối ưu như SGD không hoạt động tốt đối với dữ liệu mà số lượng dữ liệu thừa lớn. Vì các thuật toán này sử dụng thông tin của gradient để thực hiện bước cập nhật, và chính thông tin này bị mất đi hoặc có rất ít trong dữ liệu thừa. Sự thiếu hụt thông tin làm cho các thuật toán SGD hay Momentum thực hiện những bước đi rất nhỏ tại hướng tương ứng với đặc trưng này. Đó là lý do vì sao trong hình 4.7a cho ta thấy thuật toán Momentum cập nhật nhanh tại hướng tương ứng với đặc trưng đặc

b nhưng lại đi những bước rất nhỏ theo hướng của đặc trưng thừa w . Vì thực hiện những bước cập nhật lớn trên hướng b nên độ lỗi của Momentum giảm rất nhanh trong các bước đầu tiên. Tuy nhiên, độ lỗi giảm chậm tại những bước tiếp theo ứng với các bước di chuyển chậm chạp theo hướng w .

Bằng việc sử dụng tỉ lệ học riêng biệt cho từng hướng, thuật toán RMSprop và Adam tăng nhanh tốc độ cập nhật tại hướng w từ đó tăng tốc quá trình huấn luyện. Vì RMSprop và Adam đảm bảo cả hai hướng w và b đều được cập nhật nên đường đi của hai thuật toán trên bề mặt lỗi 4.7a là một đường chéo hướng thẳng về cực tiểu. Tuy nhiên Adam nhờ có thêm yếu tố momentum mà cho đường đi thẳng hơn về phía cực tiểu. Độ lỗi của thuật toán RMSprop mặc dù giảm nhanh hơn Momentum nhưng thuật toán hội tụ tại điểm có độ lỗi cao hơn. Giải thích cho hiện tượng này là do RMSprop gấp dao động tại vùng gần cực tiểu và không thể hội tụ về điểm cực tiểu. Bên cạnh đó, mặc dù Adam có tốc độ chậm hơn Momentum nhưng lại có thể hội tụ về điểm cực tiểu nhanh hơn. Một lý do là vì Adam sử dụng EMA nên cần một thời gian để có thể xấp xỉ tốt hơn. Tuy nhiên, đó không phải là nguyên nhân chính. Một lý do khác là vì bước cập nhật của Adam trên hướng có độ dốc cao b chậm hơn so với Momentum nhưng lại có những bước đi có ý nghĩa hơn Momentum theo hướng tối ưu có độ dốc thấp w . Kết quả là Adam có thể hội tụ tại cực tiểu tốt hơn và nhanh hơn Momentum.

Một khó khăn khác khi huấn luyện mạng nơ-ron là huấn luyện trên dữ liệu nhiều nhiễu. Để thực hiện quan sát cách các thuật toán xử lý nhiễu, thí nghiệm sử dụng một tập dữ liệu gồm n cặp (x, y) với y là giá trị nhãn. Sau đó, thực hiện thêm nhiễu vào trong giá trị y để cho ra \hat{y} là giá trị mà ta quan sát được. Mô hình sử dụng linear regression và hàm chi phí MSE để huấn luyện dựa trên n cặp (x, \hat{y}) . Kết quả được mô tả trong hình 4.2.4 sau 150 epochs. Ta dễ dàng nhận thấy được Momentum thực hiện bước cập nhật đầu rất nhanh nhưng sau đó lại hội tụ tại một điểm có độ lỗi cao hơn Adam và RMSprop. Ta nhận thấy trong 4.2.4b, Adam và RMSprop



Hình 4.8: Đường đi của các thuật toán trong bề mặt lỗi (trái) và độ lỗi (phải) của từng thuật toán trong trường hợp đặc trưng thừa.

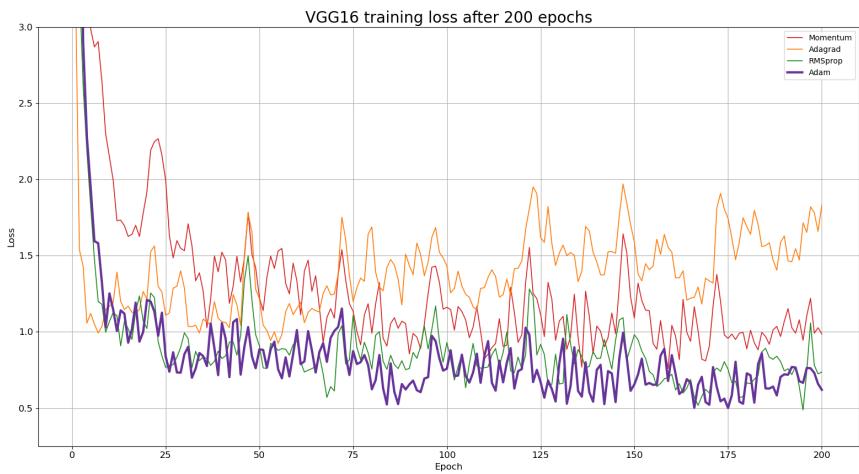
cùng hội tụ về cùng độ lỗi nhưng do Adam có thêm quan tính mà thời gian hội tụ sớm hơn. Vẫn chưa có lý do tại sao các thuật toán tỉ lệ học thích ứng lại có thể hoạt động tốt trong trường hợp này. Một bài báo của Balles Lukas và Hennig Phillip [2] cho rằng vì Adam không thực sự sử dụng kết quả của gradient mà thay vào đó xác định hướng chỉ dựa vào dấu của gradient. Chính vì vậy mà Adam không bị ảnh hưởng bởi độ nhiễu có trong dữ liệu trong khi Momentum cố gắng fit nhiễu vào trong kết quả dự đoán. Đây được coi là lý do tại sao Momentum hội tụ tại một điểm có độ lỗi cao hơn.

4.2.5 Huấn luyện mô hình VGG16

Chúng tôi thực hiện huấn luyện kiến trúc mạng tích chập nhiều tầng ẩn VGG16 của Karen Simonyan và Andrew Zisserman [25] trên tập dữ liệu ImageNet. Đây là một kiến trúc mạng nơ-ron nhiều tầng ẩn được sử dụng rộng rãi trong rất nhiều bài toán khác nhau, không chỉ riêng bài toán phân lớp do khả năng trích xuất đặc trưng rất mạnh. Kiến trúc mạng VGG16 cũng thường được sử dụng làm thước đo đánh giá độ hiệu quả của các

thuật toán huấn luyện mạng nơ-ron nhiều tầng ẩn [0][24]. Các thiết lập huấn luyện như kích thước minibatch, kích thước và quá trình tiền xử lý ảnh đầu vào được giữ nguyên như trong bài báo gốc của VGG16 [25].

Với mỗi thuật toán tối ưu, chúng tôi cố gắng dò tìm bộ siêu tham số cho kết quả tốt nhất. Tuy nhiên, do việc huấn luyện mạng VGG16 với khoảng 138 triệu tham số trên một tập dữ liệu lớn như ImageNet rất tốn kém nên chúng tôi không thể dò tìm siêu tham số một cách chi tiết như các thí nghiệm khác. Một trường hợp đặc biệt là thuật toán SGD thường như không thể tối ưu cho kiến trúc mạng này khi độ lỗi luôn giữ ở mức cao bất kể siêu tham số mà chúng tôi thử nghiệm, vì vậy chúng tôi không bao gồm biểu đồ của SGD trong kết quả.



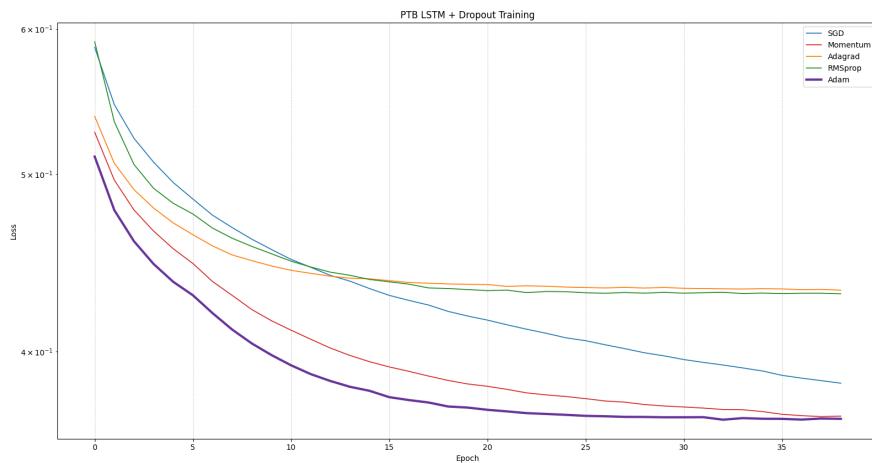
Hình 4.9: Đường đi của các thuật toán trong bề mặt lỗi (trái) và độ lỗi (phải) của từng thuật toán trong trường hợp đặc trưng thừa.

Quá trình huấn luyện mô hình VGG16 trên tập dữ liệu ImageNet (hình 4.2.5) cho thấy Adam là thuật toán tối ưu hiệu quả nhất trong số các thuật toán được thí nghiệm khi có độ lỗi giảm nhanh chóng và ổn định. Adagrad và Momentum bị dao động mạnh trong suốt quá trình huấn luyện và không đạt được độ lỗi thấp. Chỉ có RMSprop là có độ lỗi gần với Adam nhất, nhưng RMSprop cũng có biên độ dao động lớn hơn.

4.2.6 Huấn luyện mô hình ngôn ngữ

Chúng tôi sử dụng thiết lập giống trong bài báo của Zaremba và cộng sự [30] cho mô hình LSTM-medium. Sử dụng kiến trúc mạng RNN gồm 2 tầng Long Short-term Memory (LSTM), mỗi tầng có 650 nơ-ron với số bước unroll là 35. Chúng tôi khởi tạo hai tầng ẩn này bằng giá trị 0 và sử dụng trạng thái ẩn cuối cùng của minibatch t làm giá trị khởi tạo của trạng thái ẩn của minibatch $t + 1$ với kích thước của một minibatch là 20.

Các giá trị trọng số trong mạng được khởi tạo theo phân phối đều trong khoảng $[-0.05, 0.05]$. Với tỷ lệ dropout là 0.5 cho các liên kết không hồi quy, mô hình được huấn luyện trong 39 epoch với tỷ lệ học được tuning tốt nhất cho mỗi thuật toán tối ưu và được trình bày trong bảng A.3. Trong quá trình huấn luyện, sau epoch thứ 6 thì tỷ lệ học sẽ được nhân với $\frac{5}{6}$ sau mỗi epoch. Giá trị clip của gradient được sử dụng là 5.



Hình 4.10: Mô tả kết quả thí nghiệm của mô hình ngôn ngữ trên tập dữ liệu Penn Treebank gồm kết quả trên tập huấn luyện (trái) và tập kiểm thử (phải).

Hình 4.2.6 cho ta kết quả chạy của mô hình LSTM trên tập Penn Tree Bank trong 39 epoch. Từ đó, ta thấy được rằng Adam cho độ lỗi nhỏ nhất ngay từ những epoch đầu tiên, theo ngay sau đó là Momentum. Mặc dù

RMSProp cho độ lỗi giảm mạnh nhưng rất nhanh chững lại và không thể xuống điểm có độ lỗi thấp hơn nữa.

Chương 5

Kết luận và hướng phát triển

5.1 Kết luận

Placeholder

5.2 Hướng phát triển

Placeholder

Tài liệu tham khảo

Tiếng Anh

- [1] Jimmy Ba and Rich Caruana. “Do Deep Nets Really Need to be Deep?” In: *CoRR* abs/1312.6184 (2013).
- [2] Lukas Balles and Philipp Hennig. “Dissecting adam: The sign, magnitude and variance of stochastic gradients”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 404–413.
- [3] Yoshua Bengio. “Learning Deep Architectures for AI”. In: *Foundations and Trends in Machine Learning* 2.1 (2009). DOI: 10.1561/2200000006.
- [4] Yoshua Bengio and Yann Lecun. “Scaling Learning Algorithms towards AI”. In: *Large-Scale Kernel Machines*. MIT Press, 2007.
- [5] Léon Bottou. “Large-Scale Machine Learning with Stochastic Gradient Descent”. In: *Proceedings of COMPSTAT’2010*. Ed. by Yves Lechevallier and Gilbert Saporta. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186.
- [6] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. “Optimization Methods for Large-Scale Machine Learning”. In: *SIAM Review* 60.2 (2018), pp. 223–311.
- [7] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. “Deep Blue”. In: 134.1–2 (2001), 57—83.

- [8] Yann N Dauphin et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014.
- [9] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.
- [10] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *J. Mach. Learn. Res.* 12 (2011), 2121–2159.
- [11] Dumitru Erhan et al. “The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training”. In: *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*. 2009, pp. 153–160.
- [12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [13] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. “Optimal Whitenning and Decorrelation”. In: *The American Statistician* 72 (2015), pp. 309 –314.
- [14] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).
- [15] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*, tech. rep. 2009.
- [16] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/> 2 (2010).
- [17] Hao Li et al. “Visualizing the Loss Landscape of Neural Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018.

- [18] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. “Building a Large Annotated Corpus of English: The Penn Treebank”. In: *Comput. Linguist.* 19.2 (June 1993), 313–330. ISSN: 0891-2017.
- [19] Quynh Nguyen and Matthias Hein. “The Loss Surface of Deep and Wide Neural Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 2603–2612.
- [20] Michael Nielsen. *Neural Network and Deep Learning*. Determination Press, 2015.
- [21] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).
- [22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, 318–362.
- [23] Gerard Salton and Christopher Buckley. “Term-weighting approaches in automatic text retrieval”. In: *Information Processing and Management* 24.5 (1988), pp. 513–523.
- [24] Frank Schneider, Lukas Balles, and Philipp Hennig. “DeepOBS: A Deep Learning Optimizer Benchmark Suite”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=rJg6ssC5Y7>.
- [25] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (2014).
- [26] M. Staib et al. “Escaping Saddle Points with Adaptive Gradient Methods”. In: *ICML*. 2019.

- [27] Ilya Sutskever et al. “On the importance of initialization and momentum in deep learning”. In: *ICML*. 2013.
- [28] Richard Sutton. “Two problems with back propagation and other steepest descent learning procedures for networks”. In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*. 1986, pp. 823–832.
- [29] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude”. 2012.
- [30] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. “Recurrent neural network regularization”. In: *arXiv preprint arXiv:1409.2329* (2014).
- [0] Juntang Zhuang et al. “AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients”. In: *Advances in Neural Information Processing Systems* 33 (2020).

s

Phụ lục A

Các siêu tham số của các thí nghiệm

Thuật toán	Tỉ lệ học	Momentum/ $Beta_1$	Alpha/ $Beta_2$	ϵ
SGD	0.01	-	-	-
Momentum	0.01	0.9	-	-
Adagrad	0.045	-	-	1e-8
RMSprop	0.0001	-	0.91	1e-8
Adam	0.0001	0.91	0.998	1e-8

Bảng A.1: Các siêu tham số được sử dụng trong thí nghiệm Multi-layer Neural Network.

Thuật toán	Mục đích	Tỉ lệ học	Momentum/Beta₁	Alpha/Beta₂	ϵ
SGD	Tốt nhất	0.1	-	-	-
Momentum	Tốt nhất	0.00728	0.9127	-	-
Nesterov momentum	Tái tạo	0.001	0.99	-	-
Adagrad	Tái tạo	0.001	-	-	-
	Tốt nhất	0.01	-	-	-
RMSprop	Tốt nhất	0.0004338	-	0.9144	1e-8
Adam	Tái tạo	0.001	0.9	0.999965	1e-9
	Tốt nhất	0.00033	0.5	0.999935	1e-8

Bảng A.2: Các siêu tham số được sử dụng trong thí nghiệm Convolutional Neural Network.

Thuật toán	Tỉ lệ học	Momentum/Beta₁	Alpha/Beta₂	ϵ
SGD	1	-	-	-
Momentum	0.666182	0.939152	-	-
Adagrad	0.003749	-	0.9944574	1e-8
RMSprop	0.0001	-	0.91	1e-8
Adam	0.001468	0.9277077	0.998469	1e-8

Bảng A.3: Các siêu tham số được sử dụng trong thí nghiệm mô hình ngôn ngữ.