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

Nguyễn Ngọc Lan Như - Hoàng Minh Quân

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

KHÓA LUẬN TỐT NGHIỆP CỬ 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 CỬ 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

Danh mục hình ảnh

Danh mục bảng

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.

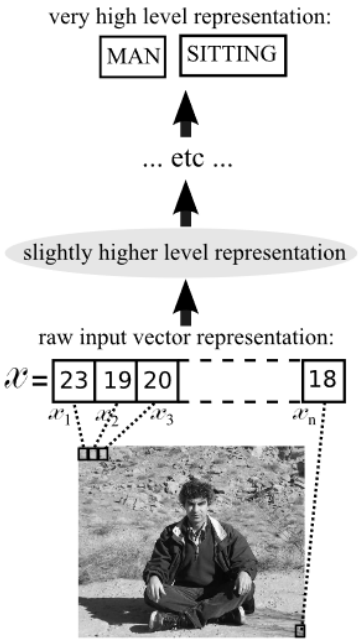
Chương 1

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 việc 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. Việc có nhiều tầng ẩn cũng là một trong những lý do giúp cho 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 [14] 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

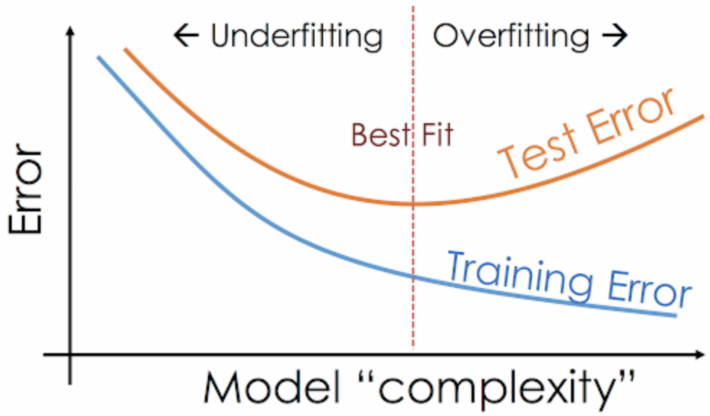
tạp hơn (higher-level features) [3]; một công trình khác của Yoshua Bengio cho thấy sức mạnh của mạng nơ-ron nhiều tầng ẩn khi nó có thể giải quyết các bài toán mà các mạng nơ-ron ít tầng ẩn hơn không thể[2].



Hình 1.1: Minh hoạ quá trình mạng nơ-ron "học" từ các đặc trưng đơn giản, như các điểm ảnh, tạo thành các đặc trưng phức tạp hơn, như câu mô tả.[2]

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.

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

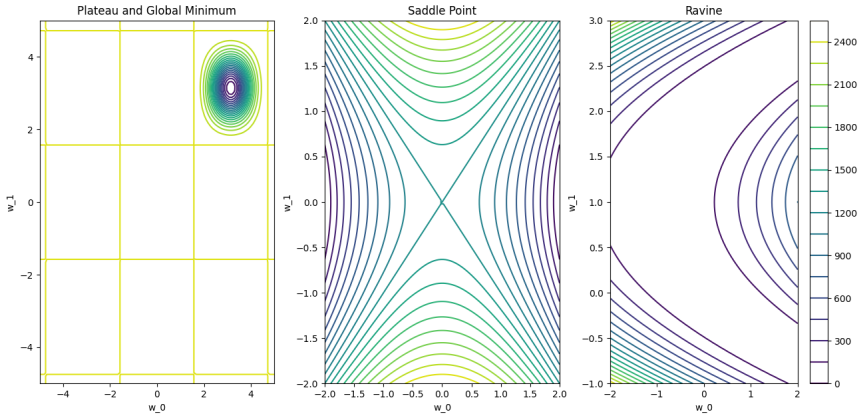


Hình 1.2: Minh họa cho mô hình overfit và underfit. (Nguồn: `https://www.analyticsvidhya.com/blog/2020/02/underfitting-overfitting-best-fitting-machine-learning`)

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.

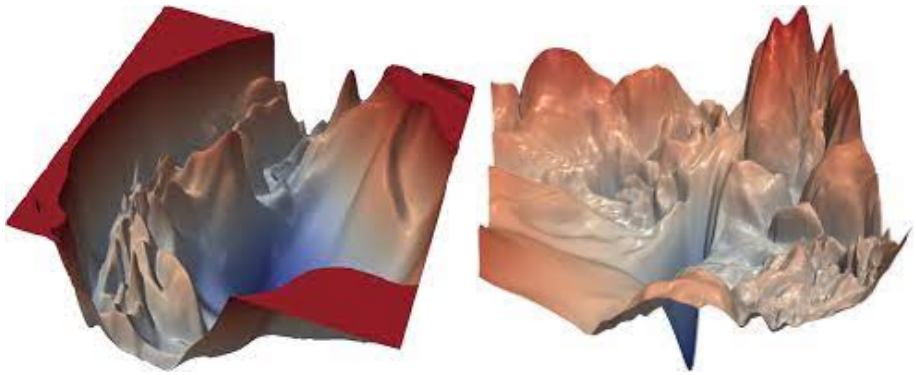
Bài toán tối ưu hoá mạng nơ-ron nhiều tầng ẩn nhận dữ liệu nhập là hàm chi phí nhận bộ trọng số của mạng nơ-ron nhiều tầng ẩn làm tham số. Hàm chi phí cho biết sự sai lệch giữa kết quả dự đoán của mạng nơ-ron so với giá trị đúng. Giá trị sai lệch này còn được gọi là độ lỗi. Sau quá trình tối ưu ta mong 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.

Việc tối ưu hoá 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 hoá mạng nơ-ron là quá trình di chuyển trong mặt phẳng lỗi để tìm được tọa độ của một điểm trên mặt phẳng được định nghĩa bởi hàm lỗi cho giá trị độ lỗi là đủ nhỏ. Tuy nhiên, việc di chuyển trong mặt phẳng lỗi gặp nhiều khó khăn. Li Hao và cộng sự [12] cho thấy rằng sự hỗn loạn của mặt phẳng lỗi ngày càng tăng khi số tầng ẩn trọng



Hình 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).

mạng nơ-ron ngày càng tăng. Sự hỗn loạn này được cấu thành từ nhiều "critical point", những điểm có gradient bằng 0, như cực tiểu địa phương, điểm yên ngựa, hợp với các vùng bằng phẳng, và vùng rãnh hẹp.



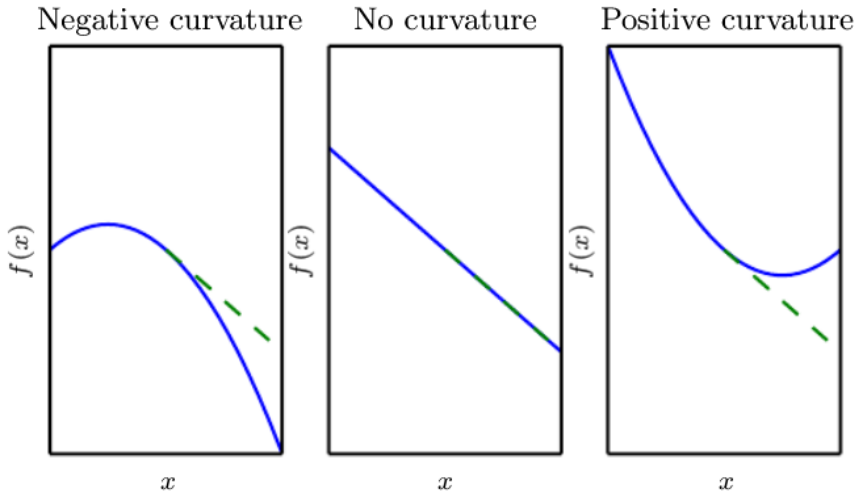
Hình 1.4: Bề mặt lỗi của mô hình ResNet-110 (trái) và bề mặt lỗi của mô hình ResNet-56 (phải).[12]

Các điểm cực tiểu địa phương từng được xem như là nguyên nhân chính gây ra sự khó khăn trong việc tối ưu hoá mạng nơ-ron có nhiều tầng ẩn, nhưng nghiên cứu của Quynh Nguyen và Matthias Hein [13] cũng như công trình của Anna Choromanska và cộng sự [7] đã chỉ ra rằng khi số lượng nơ-ron trong một tầng nhiều hơn điểm dữ liệu huấn luyện và các điểm dữ liệu độc lập tuyến tính với nhau thì hầu hết các điểm cực tiểu là cực tiểu toàn cục (hoặc có độ lỗi rất gần với cực tiểu toàn cục). Những nghiên cứu

này cũng cho rằng việc đi tìm cực tiểu toàn cục thật sự có thể mất nhiều thời gian nhưng cho hiệu quả không đáng kể. Vì thế cực tiểu địa phương không phải là nguyên nhân chính dẫn đến sự hội tụ chậm trong quá trình huấn luyện mà các điểm yên ngựa được bao bọc bởi vùng bằng phẳng mới là nguyên nhân chính dẫn đến những khó khăn trong việc tối ưu hoá mạng nơ-ron nhiều tầng ẩn. Yann N. Dauphin đã chỉ ra rằng số lượng điểm yên ngựa tăng lên theo tỷ lệ hàm mũ với số lượng tầng ẩn có trong mạng [8], khiến cho điểm yên ngựa là điểm xuất hiện nhiều nhất so với các critical point khác như cực tiểu cục bộ. Khi số lượng tầng ẩn quá lớn thì tất cả 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 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. Vì thế, việc tránh và tìm được cách thoát ra khỏi điểm yên ngựa là quan trọng đối với các thuật toán tối ưu.

Trong thời gian đầu, việc huấn luyện mạng nơ-ron nhiều tầng ẩn sử dụng các phương pháp bậc nhất, các phương pháp này chỉ sử dụng đạo hàm bậc nhất của bề mặt lỗi tại vị trí hiện tại. Việc sử dụng đạo hàm bậc nhất/gradient sẽ cho biết hướng có độ dốc lớn nhất của bề mặt lỗi tại một điểm. Do đó, các phương pháp bậc nhất thường chỉ thực hiện những bước cập nhật rất nhỏ theo hướng gradient để tránh trường hợp bị vượt qua khỏi những vùng thấp hơn có khả năng chứa cực tiểu của bề mặt lỗi. Các phương pháp bậc nhất thường không gặp vấn đề mắc kẹt tại điểm yên ngựa vì hướng của gradient luôn đảm bảo tìm thấy một điểm mới có độ lỗi nhỏ hơn điểm hiện tại (nếu điểm hiện tại không phải là critical point). Thế nhưng, chúng thường bị mắc kẹt tại các vùng bằng phẳng xung quanh điểm yên ngựa. Một số phương pháp áp dụng nguyên lý về quán tính để gia tăng tốc độ khi gặp những chiều di chuyển ổn định để nhanh chóng vượt qua vùng bằng phẳng.

Do đạo hàm bậc nhất thiếu đi thông tin về độ cong bề mặt lỗi, nên các thuật toán bậc nhất không thể tăng tốc tại các vùng có độ cong thấp

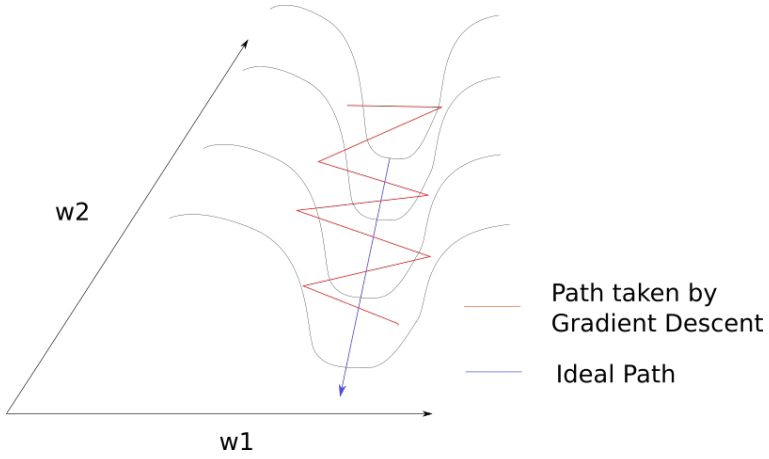


Hình 1.5: Mô tả điểm yếu của các thuật toán 1st-order không thể phân biệt đường cong xuống (trái), mặt phẳng (giữa), đường cong lên (phải).[10]

(low-curvature) và giảm tốc khi gặp vùng có độ cong cao (high-curvature) như các thuật toán sử dụng đạo hàm bậc hai. Tuy nhiên, các phương pháp bậc hai, như thuật toán Newton, thường bị các điểm cực tiểu và cực đại "thu hút" nên dễ dàng bị mắc kẹt tại các điểm yên ngựa, tại đó vừa là cực tiểu trên chiều này nhưng là cực đại trên chiều khác. Để giải quyết hiện tượng này, một số cải tiến loại bỏ hoàn toàn chiều có độ cong hướng xuống tại điểm đang xét. Tuy nhiên, cách giải quyết này không đảm bảo chiều bị loại bỏ là chiều mà tại đó điểm yên ngựa là cực đại, dẫn đến hướng ra không được xem xét và thuật toán vẫn mắc kẹt tại điểm yên ngựa. Một cách cải tiến khác là sử dụng vùng tin cậy (trust region) bằng cách cộng thêm một hệ số dampening factor α vào công thức cập nhật. Hệ số này đảm bảo rằng ma trận Hessian là một ma trận xác định dương (positive definite), tất cả phần tử trong ma trận đều không âm. Cách cải tiến này tuy có thể giúp thuật toán thoát khỏi vùng yên ngựa nhưng lại có thể mất nhiều thời gian để thoát khỏi vùng bằng phẳng nếu hệ số α quá lớn. Mặt khác, việc tính toán đạo hàm bậc hai đòi hỏi nhiều chi phí tính toán và tiêu tốn bộ nhớ cấp số mũ với số lượng trọng số của mạng. Đối với các mạng nơ-ron nhiều tầng ẩn mà số lượng trọng số có thể lên đến hàng tỷ

thì đây là bất khả thi. Vì vậy, một số phương pháp bậc nhất thực hiện xấp xỉ thông tin của đạo hàm bậc hai (cụ thể là xấp xỉ ma trận Hessian) từ gradient để thực hiện cập nhật bước nhảy tối ưu hơn cho từng trọng số. Họ các thuật toán này được gọi là các thuật toán có tỉ lệ học thích ứng (adaptive learning rate).

Nhìn chung, mỗi nhóm phương pháp cố gắng khắc phục một vấn đề của bài toán, nhưng lại không hoạt động tốt trong những trường hợp mà nhóm phương pháp còn lại có thể giải quyết được. Cụ thể là với những phương pháp sử dụng gradient kết hợp với một số phương pháp tăng tốc, việc cập nhật chỉ được thực hiện theo hướng của gradient mà không có nhiều thông tin về chiều dài mà bước cập nhật nên thực hiện, cũng như hướng của gradient không phải lúc nào cũng là hướng đi tốt nhất.



Hình 1.6: 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/`)

Một trường hợp thể hiện hướng đi của gradient không phù hợp là trường hợp di chuyển trong rãnh hẹp, hai bên có độ dốc cao và một rãnh thấp ở giữa. Khi đó, gradient sẽ hướng về phía sườn dốc còn lại thay vì hướng theo rãnh thấp, khiến cho điểm di chuyển bị dao động mạnh trong khi độ lỗi không giảm nhiều. Hiện tượng dao động này có thể được khắc phục một phần bằng quán tính, tuy nhiên chính quán tính lại có thể tạo ra một số vấn đề như đi vượt khỏi điểm cực tiểu mong muốn. Trong khi đó, những

phương pháp sử dụng tỉ lệ học thích ứng mặc dù di chuyển tốt hơn về phía điểm cực tiểu, nhưng lại thực hiện những bước cập nhật rất chậm. Ngoài ra, lợi thế của tỉ lệ học riêng biệt cho từng trọng số cũng bị suy giảm nếu như các trọng số phụ thuộc vào nhau.

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 [11]. 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 lượng cập nhật cho từng trọng số. Nhờ sự kết hợp này, Adam kế thừa được nhiều ưu điểm, đồng thời khắc phục những hạn chế của các hướng tiếp cận trên khi có thể tăng giảm độ dài bước cập nhật tùy theo điều kiện bề mặt lỗi, cũng như điều chỉnh tỉ lệ học thích ứng cho từng trọng số của mạng nơ-ron. 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

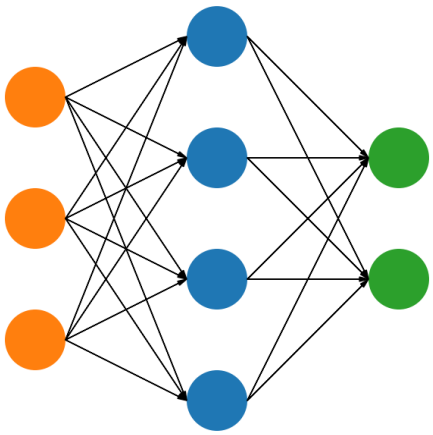
Kiến thức nền tảng

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)[6]. 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 học và 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 học máy có thể dùng để ứng dụng trong nhiều lĩnh vực khác nhau.

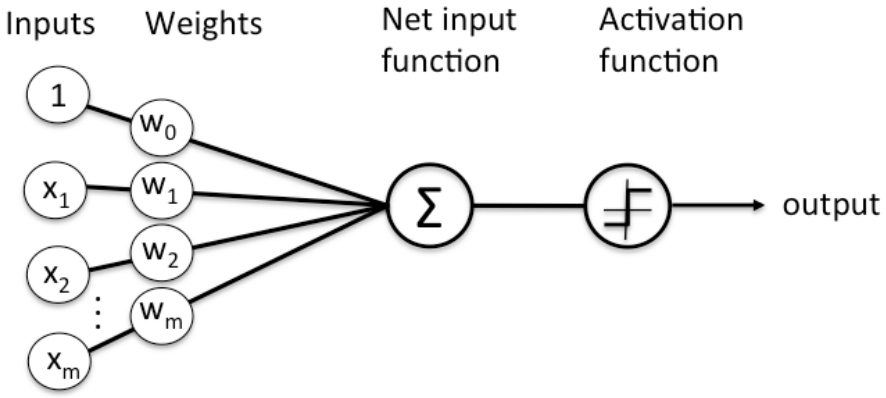


Hình 2.1: Mô hình mạng nơ-ron nhân tạo với tầng nhập (màu cam), tầng ẩn (màu xanh dương) và tầng xuất (màu xanh lá).

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 biến số biểu diễn tốt các hàm này. 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.

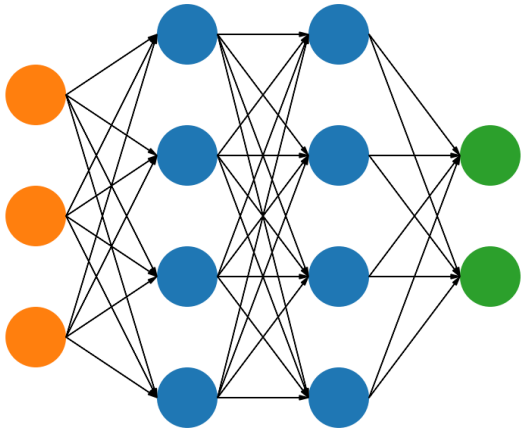


Hình 2.2: Minh họa cách hoạt động của một nơ-ron. (Nguồn: `https://wiki.pathmind.com/neural-network`)

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[3].

"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 no-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.[10]

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 huyện 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:

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

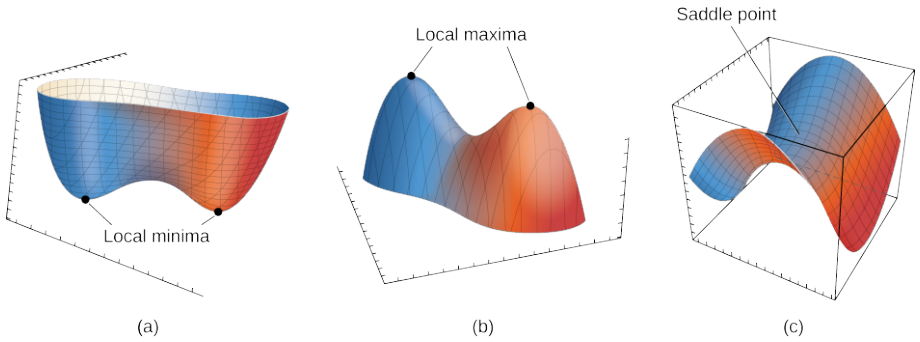
$$\Rightarrow E(\theta) = \frac{1}{N} \cdot \sum_{i=1}^{N} \mathcal{L}(\mathcal{F}_\theta(x_i), y_i) \tag{2.2}$$

Trong đó, θ là bộ trọng số của mô hình, $x_i = \{x_1, x_2, ...x_N\}$ 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 cực tiểu 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 tới 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[9] 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].



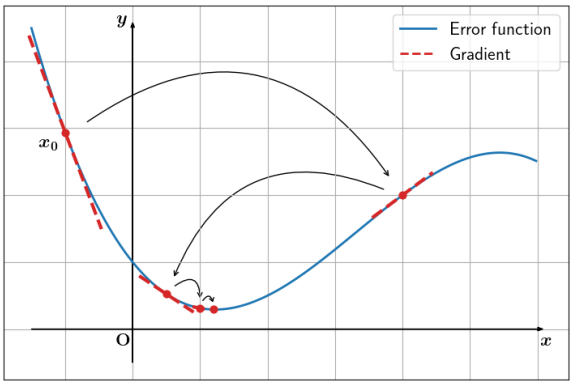
Hình 2.5: Minh hoạ điểm cực tiểu (a), cực đại (b) và điểm yên ngựa (c) trong không gian ba chiều.

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 dường 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 xannh 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.

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

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:

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

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 $f(x_1, x_2, x_3, ..., x_n)$ 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 $f(x_1, x_2, x_3, ..., x_n)$, cụ thể là:

$$\nabla f = (\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, \frac{\delta f}{\delta x_3}, ..., \frac{\delta f}{x_n}) \tag{2.5}$$

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:

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

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 khỏi 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 2.6 và (9), ta có thể viết được công thức cập nhật của θ tại mỗi bước:

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

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.

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

Thuật toán 1 Batch Gradient Descent (BGD)

Đầu vào: Tập dữ liệu huấn luyện x_i ($i = 1, 2, ..., N$), 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

$\nabla_\theta E(\theta)$, 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 (6) và (7) của BGD, chúng ta sẽ có công thức tính độ lỗi của MGD với một tập con:

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

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 (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 lân 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[10].

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.

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[4][5]. 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 hật 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 Batch Gradient Descent (BGD)

Đầu vào: Tập dữ liệu huấn luyện x_i ($i = 1, 2, ..., N$), 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

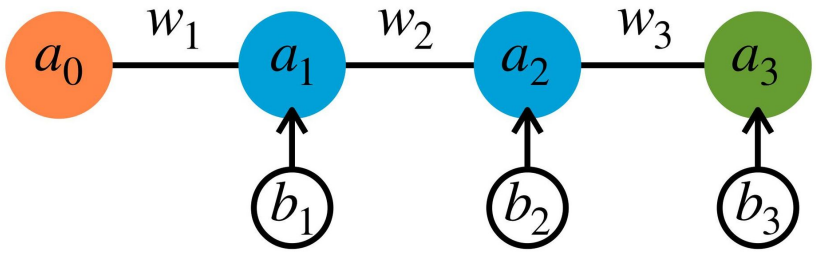
2.4 "Lan truyền ngược" (Backpropagation)

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[15]. 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 biên 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: $\frac{\delta E}{\delta w_1}; \frac{\delta E}{\delta w_2}; ...; \frac{\delta E}{\delta w_n}$. 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 2.7 với 1 tầng nhập, 2 tầng ẩn và 1 tầng xuất, mỗi tầng chỉ có một nơ-ron.



Hình 2.7: 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.

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:

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

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

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

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:

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

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

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

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

Các công thức 2.11, 2.12 và 2.12 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 $z_j = ... + w_{jk}a_k^{l-1} + ... + b_j$, ta có:

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

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.

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

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

3.1 Thuật toán Gradient Descent với Momentum

Placeholder

3.2 Thuật toán Gradient Descent với tỉ lệ học thích ứng

Placeholder

3.3 Thuật toán Adam

Placeholder

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 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. 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.

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. 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 trên nền tảng Google Colaboratory.

4.2 Các kết quả thí nghiệm

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

Placeholder

4.2.2 Phân tích trường hợp bề mặt lỗi align và không align với tham số

Placeholder

4.2.3 Vấn đề dữ liệu thưa và nhiều lỗi

Placeholder

4.2.4 Thử nghiệm trên mô hình VGG16

Placeholder

4.2.5 Thử nghiệm trên mô hình RNN

Placeholder

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] Yoshua Bengio. "Learning Deep Architectures for AI". In: *Foundations and Trends in Machine Learning* 2.1 (2009). DOI: 10.1561/2200000006.

[3] Yoshua Bengio and Yann Lecun. "Scaling Learning Algorithms towards AI". In: *Large-Scale Kernel Machines*. MIT Press, 2007.

[4] 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.

[5] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. "Optimization Methods for Large-Scale Machine Learning". In: *SIAM Review* 60.2 (2018), pp. 223–311.

[6] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. "Deep Blue". In: 134.1–2 (2001), 57–83.

[7] Anna Choromanska et al. "The Loss Surface of Multilayer Networks". In: *CoRR* abs/1412.0233 (2014).

[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] D. Erhan et al. “The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training”. In: *AISTATS*. 2009.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[12] 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.

[13] 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.

[14] Michael Nielsen. *Neural Network and Deep Learning*. Determination Press, 2015.

[15] 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.