TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ
NHẬP MÔN HỌC MÁY

Sinh viên:
Nguyễn Trọng Đạt - 52100176
Giảng viên hướng dẫn:
PGS. TS. Lê Anh Cường

TP. Hồ Chí Minh, 2023.

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ
NHẬP MÔN HỌC MÁY

Giảng viên hướng dẫn:
PGS. TS. Lê Anh Cường
Sinh viên:
Nguyễn Trọng Đạt - 52100176

TP. Hồ Chí Minh, 2023.

LỜI CẢM ƠN

Trước hết chúng em xin gửi lời cảm ơn đến PGS. TS. Lê Anh Cường, người đã hướng dẫn em trong quá trình em học môn **Nhập môn học máy**. Sự hướng dẫn của thầy đã giúp em có thêm những kiến thức về môn học và trang bị những kiến thức cần thiết để làm đồ án này. Qua những chỉ dẫn của thầy giúp em hiểu sâu hơn về những kiến thức đã được học.

Em gửi lời cảm ơn đến các bạn trong lớp, những người đã luôn ở bên giúp đỡ, hỗ trợ em để em hoàn thành tiểu luận này.

Và em xin gửi lời cảm ơn đến gia đình, bạn bè đã tạo mọi điều kiện để em xây dựng thành công đồ án này.

Trường ĐH Tôn Đức Thắng,
Ngày 24 tháng 12 năm 2023.
Sinh viên thực hiện,
(Ký và ghi rõ họ tên)

Nguyễn Trọng Đạt.

BÀI TIỂU LUẬN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Em xin cam đoan đây là bài báo cáo sản phẩm Báo cáo cuối kì của chỉ riêng của em. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa được công bố dưới bất kỳ hình thức nào. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đề tài còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc rõ ràng và cụ thể.

Nếu phát hiện có bất kỳ sự gian lận nào em xin hoàn toàn chịu trách nhiệm về nội dung của bài Báo cáo cuối kì. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền trong quá trình thực hiện của em.

Trường ĐH Tôn Đức Thắng,
Ngày 24 tháng 12 năm 2023.
Sinh viên thực hiện,
(Ký và ghi rõ họ tên)

Nguyễn Trọng Đạt.

TÓM TẮT

Cấu trúc báo cáo được chia thành 10 phần chính

- Chương 1: Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy
  Bao gồm tìm hiểu các phương thức Optimizer và so sánh chúng.
- Chương 2: Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó
  Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy

Mục lục

Danh sách hình vẽ

THUẬT NGỮ VIẾT TẮT

- GD: Gradient Descent
- BCD: Batch Gradient Descent
- SGD: Stochastic Gradient Descent
- NAG: Nesterow Accelerated Gradient

Chương 1

Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy

1.1 Tìm hiểu các phương pháp Optimizer trong huấn luyện mô hình học máy

Thuật toán Optimizer trong huấn luyện mô hình học máy là thuật toán được sử dụng để cập nhật các tham số của mô hình trong quá trình đào tạo. Mục tiêu của thuật toán Optimizer là tìm ra tập hợp các tham số tối ưu hóa trong hàm mất mát (the loss function) của mô hình học.

Trong học máy, hàm mất mát (the loss function) là một hàm đo lường mức độ tốt của mô hình học trong việc dự đoán dữ liệu. Mục tiêu của việc đào tạo mô hình học máy là tìm ra tập hợp các tham số khiến hàm mất mát (the loss function) đạt giá trị nhỏ nhất.

Trong huấn luyện mô hình học máy, có nhiều thuật toán Optimizer khác nhau được sử dụng. Trong đó một số thuật toán Optimizer phổ biến như:

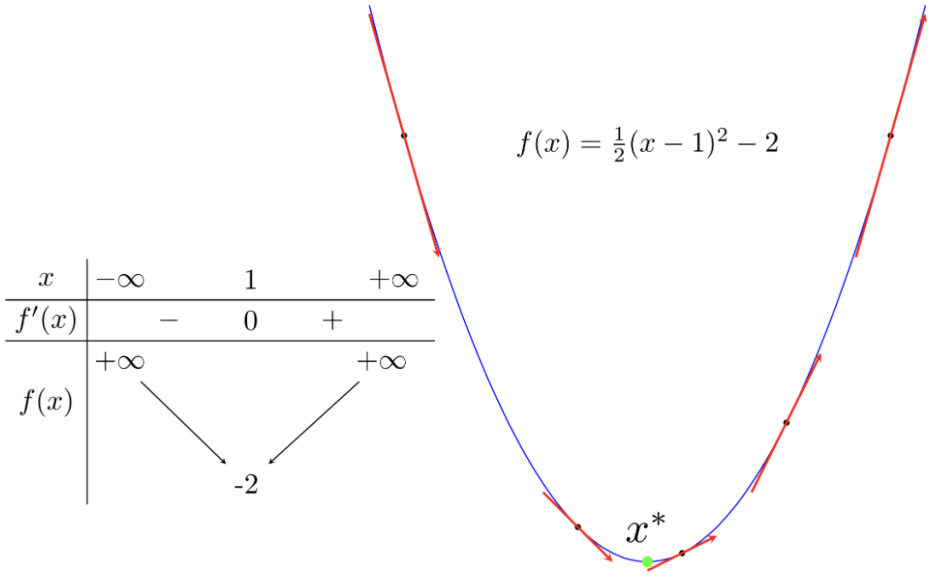
- **Gradient descent**: Đây là thuật toán Optimizer cơ bản nhất, cập nhật các tham số theo hướng ngược lại với gradient của hàm mất mát (the loss function).
- **Stochastic gradient descent (SGD)**: Đây là một biến thể của Gradient descent, cập nhật các tham số theo hướng gradient của hàm mất mát (theo loss function) tại một điểm dữ liệu ngẫu nhiên.
- **Momentum**: Một biến thể của SGD, sử dụng một lượng nhớ để giúp thuật toán vượt qua các điểm cực tiểu cục bộ.
- **Adagrad**: Một biến thể của SGD, điều chỉnh tốc độ học tập của thuật toán dựa trên độ dốc của hàm mất mát.
- **RMSProp**: Một biến thể của Adagrad, giảm thiểu hiện tượng rung lắc của tốc độ học tập.
- **Adam**: Một biến thể của RMSProp, kết hợp các ưu điểm của Momentum và RMSProp.

1.1.1 Gradient Descent

Gradient Descent là một trong những thuật toán phổ biến nhất để thực hiện việc tối ưu hóa trong quá trình học máy, và là cách tốt nhất để tối ưu hóa các mạng Neural Network hiện nay. CÓ nhiều thuật toán tối ưu khác dựa trên GD đã được đưa ra như Momentum, Adagrad, Adam,...

1.1.1.1 Ý tưởng cho thuật toán

Ở đây, ta sẽ xét 1 bài toán với hàm 1 biến với hàm số là $f(x) = 12(x-1)^2 - 2$ có bảng biến thiên và đồ thị như hình bên dưới. Ta có điểm x^* là local minimun (cực tiểu) của hàm số $f'(x^*) = 0$. Hơn thế nữa, trong lân cận của nó, đạo hàm của các điểm phía bên trái x^* là không dương, đạo hàm của các điểm phía bên phải x^* là không âm. Đường tiếp tuyến với đồ thị hàm số đó tại 1 điểm bất kỳ có hệ số góc chính bằng đạo hàm của hàm số tại điểm đó. Giả sử x_t là điểm ta tìm được sau vòng lặp thứ t. Ta cần tìm một thuật toán để đưa x_t về càng gần x^* càng tốt.



Hình 1.1: Minh hoạ hàm số $f(x) = 12(x-1)^2 - 2$

Từ hình vẽ, chúng ta có quan sát sau:

- Nếu đạo hàm của hàm số tại x_t: $f(x_t) > 0$ thì x_t nằm về phía bên phải so với x^* (và ngược lại). Để điểm tiếp theo x_{t+1} gần với x^* hơn, chúng ta cần di chuyển x_t về phía bên trái, tức là về phía âm. Hay nói cách khác, chúng ta cần di chuyển ngược dấu với đạo hàm:

$$x_t + 1 = x_t + \Delta$$

Trong đó Δ là một đại lượng ngược dấu với đạo hàm $f'(x_t)$

- x_t càng xa x^* về phía bên phải thì $f'(x_t)$ càng lớn hơn 0 (và ngược lại). Vậy, lượng di chuyển Δ, một cách trực quan, là tỉ lệ thuận với $-f'(x_t)$.

Từ nhận xét trên, chúng ta có 1 cập nhật đơn giản là:

$$x_{t+1} = x_t - \eta f'(x_t)$$

Trong đó η là một số dương được gọi là learning rate (tốc độ học). Dấu trừ thể hiện việc chúng ta phải đi ngược dấu đạo hàm (Đây cũng chính là lý do phương pháp này được gọi là Gradient Descent - đi ngược đạo hàm).

1.1.1.2 Batch Gradient Descent

Gradient Descent (GD) là một thuật toán tối ưu hóa thường được sử dụng để tìm giá trị tối thiểu toàn cục cho một hàm nhiều biến $f(\theta)$, trong đó θ là một vector biểu diễn tập hợp các tham số của mô hình cần tối ưu. Đạo hàm của hàm số tại một điểm θ bất kỳ được ký hiệu là $\nabla_\theta f(\theta)$.

Tương tự như trường hợp đơn biến, thuật toán GD cho hàm nhiều biến bắt đầu với một ước lượng ban đầu θ_0, và tại mỗi vòng lặp t, quy tắc cập nhật tham số được đưa ra bởi:

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

Ở đây:

- θ là vector tham số.
- η là tốc độ học.
- $\nabla_\theta f(\theta)$ là gradient của hàm mất mát theo vector tham số θ.

Đối với GD theo cụm dữ liệu (BGD), toàn bộ tập dữ liệu được sử dụng để tính toán gradient cho mỗi cập nhật. Tuy nhiên, BGD có thể chậm và tốn kém tính toán, đặc biệt là khi toàn bộ tập dữ liệu không thể lưu vào bộ nhớ. Nó cũng không phù hợp cho việc học trực tuyến, nơi cập nhật được thực hiện khi có dữ liệu mới.

Trong thực tế, thường sử dụng một thuật toán hiệu quả hơn được gọi là Stochastic Gradient Descent (SGD).

1.1.1.3 Stochastic Gradient Descent

Trong thuật toán Stochastic Gradient Descent (SGD), tại mỗi thời điểm, đạo hàm của hàm mục tiêu chỉ được tính dựa trên một điểm dữ liệu x_i, sau đó thực hiện cập nhật θ dựa trên đạo hàm này. Quá trình này được lặp lại cho từng điểm trên toàn bộ dữ liệu và gọi là một epoch. Với SGD, mỗi epoch ứng với N lần cập nhật θ, với N là số điểm dữ liệu.

Mặc dù cập nhật từng điểm một có thể làm giảm tốc độ của mỗi epoch, nhưng SGD yêu cầu ít epoch hơn (thường là chỉ 10 lần đầu tiên và sau đó chỉ cần một epoch khi có dữ liệu mới). Do đó, SGD thích hợp cho các bài toán với cơ sở dữ liệu lớn và yêu cầu mô hình thay đổi liên tục, tức là học trực tuyến (online learning).

Sau mỗi epoch, quan trọng là chúng ta cần phải xáo trộn thứ tự của các điểm dữ liệu để đảm bảo tính ngẫu nhiên. Điều này ảnh hưởng đến hiệu năng của SGD.

Tóm lại, SGD cập nhật tham số cho từng mẫu huấn luyện x_i với nhãn y_i như sau:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x_i; y_i)$$

Trong khi Batch Gradient Descent cho thấy đồ thị hội tụ mượt dần, SGD giúp chúng ta có thể nhanh chóng tiến đến một tham số tốt hơn. Tuy nhiên, nó có thể làm phức tạp hóa quá trình hội tụ chính xác, vì SGD có thể vượt qua các điểm cực tiểu. Mặc dù đồ thị hội tụ của nó không ổn định, nghiên cứu đã chỉ ra rằng khi giảm dần tốc độ học, SGD thường hội tụ nhanh hơn Batch Gradient Descent, gần như chắc chắn hội tụ đến một điểm cực tiểu cục bộ hoặc toàn cục để tối ưu hóa hàm mục tiêu.

1.1.1.4 Mini-Batch Gradient Descent

Khác với Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent sử dụng một số lượng n dữ liệu lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu N rất nhiều). Tương tự như SGD, Mini-Batch Gradient Descent bắt đầu mỗi epoch bằng cách xáo trộn ngẫu nhiên dữ liệu và chia dữ liệu thành các mini-batch, mỗi mini-batch chứa n điểm dữ liệu (trừ mini-batch cuối cùng có thể ít hơn n nếu N không chia hết cho n). Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm và cập nhật tham số. Công thức có thể được viết dưới dạng:

$$\theta = \theta - \eta \cdot J(\theta; x_{i:i+n}; y_{i:i+n})$$

Ở đây, $x_{i:i+n}$ được hiểu là dữ liệu từ thứ i đến thứ $i + n - 1$ (theo ký hiệu của Python). Dữ liệu này cần được xáo trộn sau mỗi epoch. Các thuật toán khác

cho Gradient Descent như Momentum, Adagrad, Adadelta, và các thuật toán khác mà chúng ta sẽ tìm hiểu trong các phần sau, cũng có thể áp dụng vào Mini-Batch Gradient Descent.

Mini-Batch Gradient Descent thường được sử dụng trong hầu hết các thuật toán Machine Learning, đặc biệt là trong Deep Learning. Giá trị n thường được lựa chọn khoảng từ 50 đến 256.

Nhìn chung, Gradient Descent giúp giảm phương sai của cập nhật tham số (đạt được sự hội tụ ổn định hơn) so với SGD.

1.1.2 Momentum

Trong Gradient Descent (GD), có thể thấy nghiệm thường rơi vào điểm cực tiểu địa phương khi bề mặt của hàm mất mát có nhiều thung lũng. Điều này dẫn đến vấn đề khi chúng ta muốn tránh các điểm cực tiểu địa phương không mong muốn.

Một giải pháp để khắc phục vấn đề này là sử dụng thuật toán Momentum. Nó được mô phỏng như việc đẩy một quả bóng xuống thung lũng. Quả bóng này tích luỹ vận tốc khi lăn xuống và ngày càng nhanh hơn trên đường đi.

Trong GD, chúng ta cần tính lượng thay đổi ở thời điểm t để cập nhật vị trí mới cho nghiệm (tức hòn bi). Nếu chúng ta coi đại lượng này như vận tốc vt trong vậy lý, vị trí mới của hòn bi sẽ là $\theta_t + 1 = \theta_t - v_t$ Dấu trừ thể hiện việc phải di chuyển ngược với đạo hàm. Công việc của chúng ta bây giờ là tính đại lượng vt sao cho nó vừa mang thông tin của đạo hàm, vừa mang thông tin của đà (tức vận tốc trước đó $v_t - 1$, chúng ta coi vận tốc ban đầu $v_0 = 0$.Một cách đơn giản nhất, ta có thể cộng (có trọng số) hai đại lượng này:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

Trong đó:

- v_t là vận tốc tại thời điểm t,
- γ là hệ số momentum (thường được chọn khoảng 0.9),
- η là tốc độ học (learning rate),
- $\nabla_\theta J(\theta)$ là gradient của hàm mất mát theo θ.

Sau đó vị trí mới của hòn bi được xác định như sau:

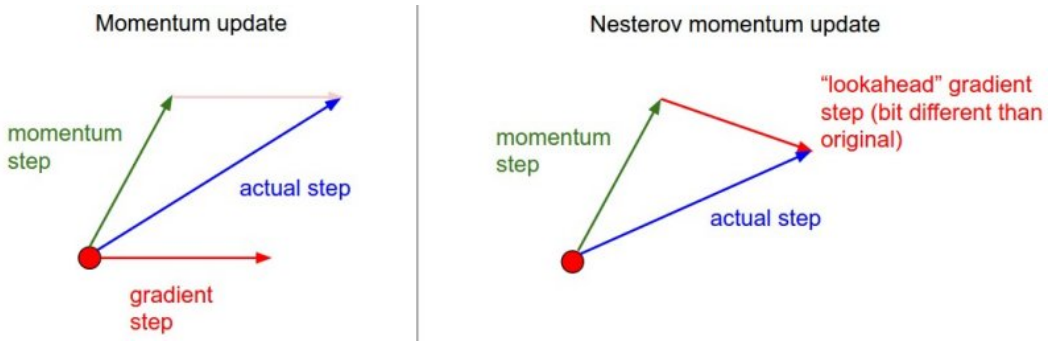
$$\theta = \theta - v_t$$

Tóm lại chúng ta có công thức đầy đủ của momentum như sau:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Công thức này giúp tận dụng thông tin của đạo hàm và đà trước đó để cập nhật vị trí mới của nghiệm. Hiệu suất của thuật toán Momentum thường tốt hơn so với Gradient Descent thông thường khi đối mặt với thung lũng và điểm cực tiểu địa phương không mong muốn.

1.1.3 Nesterow Accelerated Gradient



Hình 1.2: Ý tưởng của Nesterov Accelerated Gradient

Momentum giúp *hòn bi* (biểu diễn cho véc-tơ tham số trong thuật toán tối ưu hóa) vượt qua các đỉnh đồng địa phương, tuy nhiên, một hạn chế của nó là khi tiến gần tới đích, *hòn bi* vẫn tiếp tục di chuyển một khoảng đáng kể trước khi dừng lại. Điều này xuất phát từ việc *hòn bi* giữ lại đà từ các bước trước đó. *Nesterov Accelerated Gradient* (NAG) được thiết kế để khắc phục vấn đề này để giúp thuật toán hội tụ nhanh hơn.

Ý tưởng cơ bản của NAG là dự đoán hướng di chuyển trong tương lai bằng cách nhìn trước một bước. Cụ thể, nếu chúng ta sử dụng thành phần đà γv_{t-1} để cập nhật, chúng ta có thể ước lượng được vị trí tiếp theo của *hòn bi* là $\theta - \gamma v_{t-1}$ (ở đây chúng ta không bao gồm thành phần gradient vì chúng ta sẽ sử dụng nó ở bước kế tiếp). Do đó, thay vì sử dụng gradient tại điểm hiện tại, NAG thực hiện dự đoán cho bước tiếp theo.

Qua theo dõi hình 1.2:

- Đối với Momentum thông thường: thay đổi được tính bằng tổng của hai vector - vector đà và gradient tại thời điểm hiện tại.
- Đối với Nesterov Momentum: thay đổi được tính bằng tổng của hai vector - vector đà và gradient được ước lượng tại điểm tiếp theo.

Công thức cập nhật của NAG có thể được biểu diễn như sau:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

Ở đây:

- v_t là vector đà tại thời điểm t,
- γ là hệ số đà,
- η là tốc độ học,
- $\nabla_\theta J$ là gradient của hàm mất mát J theo tham số θ,
- θ là vector tham số cần tối ưu.

1.1.4 Adagrad

Như đã thảo luận ở mục về các vấn đề của gradient descent, thuật toán gradient descent thường áp dụng một learning rate đồng nhất cho tất cả các tham số. Tuy nhiên, trong thực tế, có những tham số ít xuất hiện và yêu cầu bước cập nhật lớn hơn (high learning rate), trong khi các tham số thường xuyên xuất hiện đòi hỏi bước cập nhật nhỏ hơn (low learning rate). Điều này thường xuyên xuất hiện trong các bài toán dữ liệu thưa (sparse). Adagrad ra đời để giải quyết vấn đề này bằng cách điều chỉnh learning rate cho từng tham số. Pennington và đồng nghiệp đã sử dụng Adagrad để huấn luyện Glove (một mô hình word embedding trong NLP), và đã đạt được hiệu suất tốt hơn so với gradient descent.

Trước đây, chúng ta thực hiện một cập nhật cho tất cả các tham số θ, cùng một lúc mỗi tham số θ_i sử dụng một learning rate η. Adagrad giải quyết vấn đề này bằng cách áp dụng learning rate khác nhau cho mỗi tham số θ_i tại mỗi bước t.

Gọi $g_{t,i}$ là gradient tại bước t của tham số θ_i. Công thức cập nhật của gradient descent cho mỗi tham số θ_i tại bước t trở thành:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

Trong quy tắc cập nhật của mình, Adagrad sửa đổi learning rate cho mỗi bước t và mỗi tham số θ_i dựa trên tổng bình phương của các gradient quá khứ đã được tính toán cho θ_i: $\sqrt{G_{t,ii} + \epsilon}$. Ở đây, G_t là ma trận đường chéo với các phần tử trên đường chéo là tổng bình phương của các gradient theo θ_i cho đến bước t, và ϵ là hệ số để tránh chia cho 0 (thường là 10^{-8}).

Công thức cập nhật tổng quát của Adagrad có thể được biểu diễn như sau bằng LaTeX:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

Thông thường, giá trị η được đặt là 0.01. Lợi ích của Adagrad là khả năng điều chỉnh learning rate cho từng tham số. Tuy nhiên, điểm yếu chính của Adagrad là tích lũy quá mức các gradient trong mẫu số, d learning rate dần giảm và trở nên cực kỳ nhỏ, khiến thuật toán không thể học được nữa. Các thuật toán sau này đã được phát triển để giải quyết vấn đề này.

1.1.5 Adadelta

Adadelta là một mở rộng của Adagrad được thiết kế để khắc phục hạn chế về việc giảm dần tốc độ học. Thay vì tích luỹ gradient bình phương của quá khứ, Adadelta giới hạn việc tích luỹ này trong một cửa sổ có kích thước cố định, được ký hiệu là w.

Để làm điều này, nó giới thiệu khái niệm trung bình trượt sử dụng mối quan hệ đệ quy sau:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

Ở đây, γ thường được đặt là 0.9.

Gọi $\Delta\theta_t$ là sự cập nhật tham số tại bước t. Đối với gradient descent (GD), sự cập nhật được xác định bởi:

$$\Delta\theta_t = -\eta \cdot g_t, \quad \theta_{t+1} = \theta_t + \Delta\theta_t$$

Tương tự, đối với Adagrad, sự cập nhật là:

$$\Delta\theta_t = -\frac{\sqrt{\eta} \odot g_t}{\sqrt{G_t} + \epsilon}$$

Bây giờ, thay thế ma trận đường chéo G_t bằng $E[g^2]_t$, sự cập nhật trở thành:

$$\Delta\theta_t = -\frac{\sqrt{E[g^2]_t + \epsilon}}{g_t}$$

Ý tưởng tiếp theo liên quan đến việc sử dụng sự cập nhật đã được hiệu chỉnh đơn vị, lấy cảm hứng từ thuật toán Newton. Sự cập nhật sau đó được xác định bởi:

$$\Delta\theta = H^{-1} g_t$$

Ở đây, H^{-1} là nghịch đảo của ma trận Hessian (đạo hàm bậc 2 của hàm mất mát). Do đó, chúng ta có:

$$\Delta\theta \propto H^{-1} g \propto \frac{\partial f}{\partial \theta} \frac{\partial^2 f}{\partial \theta^2}$$

Từ đây, chúng ta có thể xấp xỉ $\Delta\theta$ như là đạo hàm đầu tiên $\frac{\partial f}{\partial \theta}$, xem $\Delta\theta_{t-1}$ là không biết và xấp xỉ nó bằng $\text{RMS}[\theta]_{t-1}$. Tốc độ học trong biểu thức gốc được thay thế bằng $\text{RMS}[\Delta\theta]_{t-1}$, cho ra quy tắc cập nhật Adadelta như sau:

$$\Delta\theta_t = -\frac{\text{RMS}[\Delta\theta]_{t-1} \cdot g_t}{\text{RMS}[g]_t}, \quad \theta_{t+1} = \theta_t + \Delta\theta_t$$

Khởi tạo $\Delta\theta_0 = 0$, Adadelta loại bỏ nhu cầu thiết lập một tốc độ học mặc định.

1.1.6 RMSProp

RMSProp là một phương pháp điều chỉnh tốc độ học được đưa ra bởi Geoff Hinton. RMSProp và Adadelta phát triển độc lập cùng lúc xuất phát từ nhu cầu giải quyết triệt để tốc độ học giảm dần của Adagrad. RMSProp có công thức cập nhật như sau:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Ở đây, η là tốc độ học, $E[g^2]_t$ là trung bình của bình phương gradient, ϵ là một giá trị nhỏ để tránh chia cho 0.

Hinton đề xuất $\gamma = 0.9$ và giá trị mặc định tốt cho $\eta = 0.001$.

1.1.7 Adam

Adam (Adaptive Moment Estimation) là một thuật toán cho phép tính tốc độ học thích ứng với mỗi trọng số. Adam không chỉ lưu trữ trung bình bình phương các gradient trước đó như Adadelta mà còn lưu cả giá trị trung bình mô-men m_t. Các giá trị m_t và v_t được tính bởi công thức:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

trong đó β_1 và β_2 là các trọng số không âm, thường được chọn là $\beta_1 = 0.9$ và $\beta_2 = 0.999$.

Nếu khởi tạo m_t và v_t là các vector 0, các giá trị này có khuynh hướng nghiêng về 0, đặc biệt là khi β_1 và β_2 xấp xỉ bằng 1. Do đó, để khắc phục, các giá trị này được ước lượng bằng cách:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

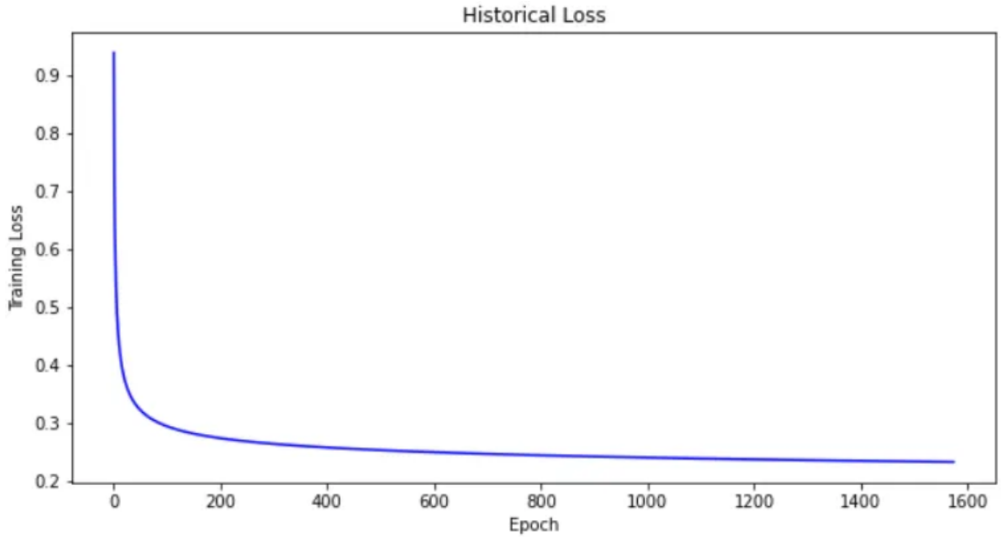
Sau đó cập nhật các trọng số theo công thức:

$$\theta_t = \theta_{t-1} - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

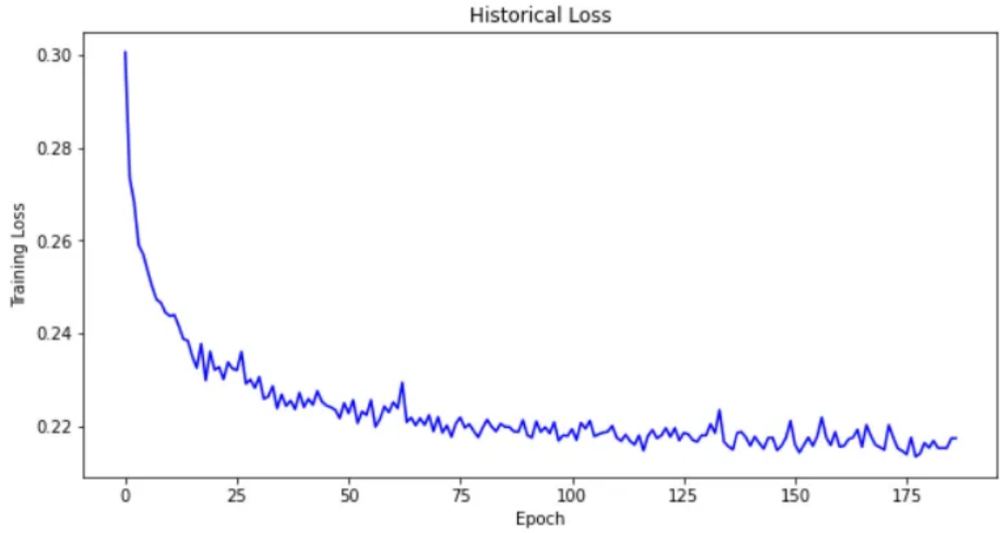
ϵ thường bằng 10^{-8}.

1.2 So sánh các phương pháp Optimizer trong huấn luyện mô hình học máy

- Độ hội tụ (Convergence)

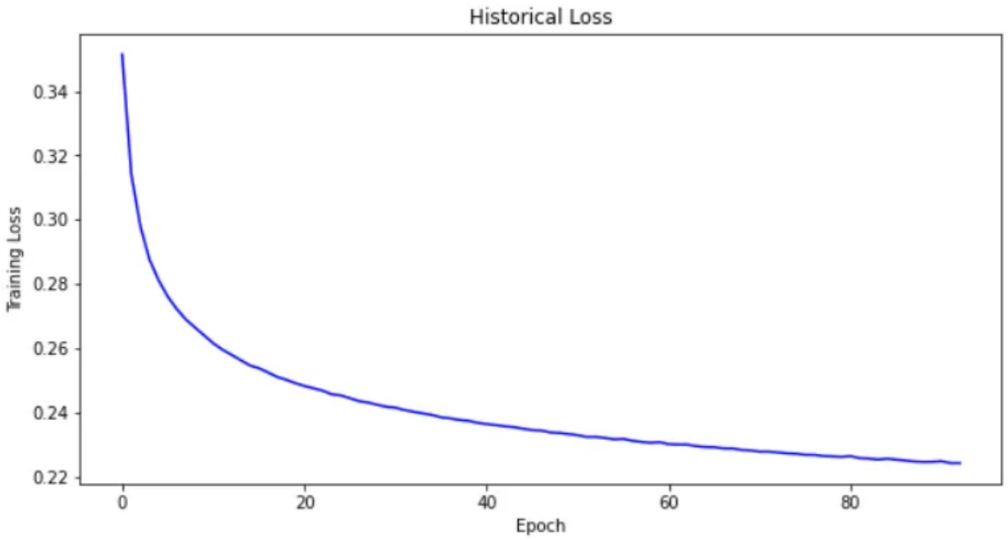


Hình 1.3: Batch gradient descent: Loss versus epoch



Hình 1.4: Stochastic gradient descent: Loss versus epoch

Ta có thể thấy, batch gradient descent có đường hội tụ ổn định trong khi stochastic gradient descent có độ nhiễu khá mạnh và mini-batch grandient descent có



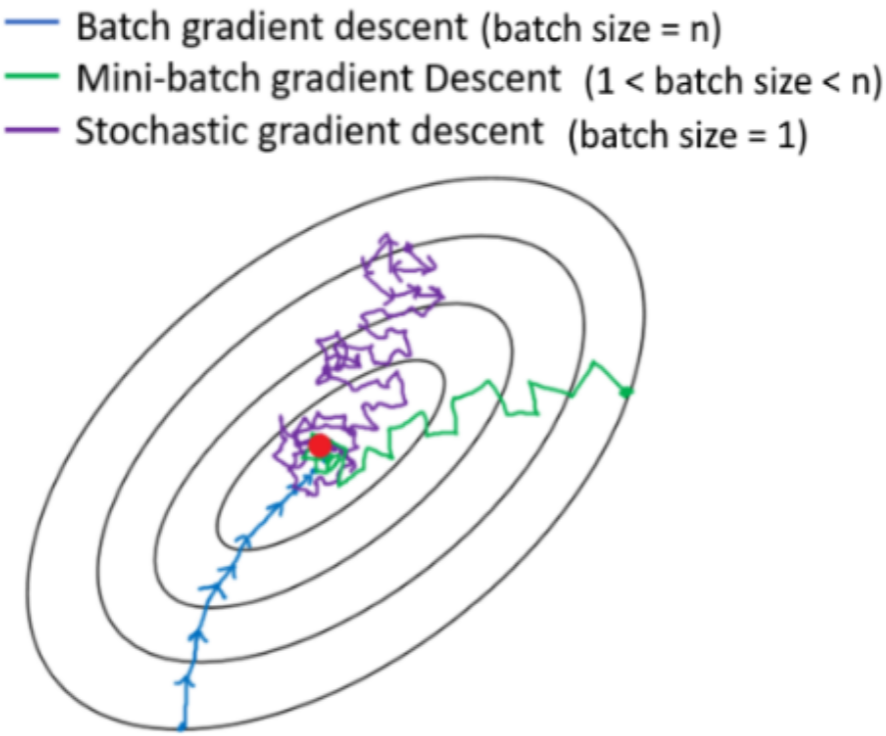
Hình 1.5: Mini-batch gradient descent: Loss versus epoch

độ nhiễu tương đối không đáng kể.
Thời gian hội tụ trong quá trình huấn luyện là:

- Batch gradient descent: 727.65 seconds
- Stochastic gradient descent: 544.12 seconds
- Mini-batch gradient descent: 45.66 seconds

Đối với stochastic gradient descent, có mỗi điểm dữ liệu một lần cập nhật và ví dụ trong trường hợp, có 60000 dữ liệu huấn luyện, thì sẽ có 60000 bản cập nhật mỗi lần lặp. Do đó, phương pháp stochastic gradient descent có thể đạt được sự hội tụ tương đối nhanh hơn với độ nhiễu lớn hơn so với phương pháp batch gradient descent.
Còn đối với mini-batch gradient descent, mỗi b điểm dữ liệu là một lần cập nhật. Giả sử chọn b=64, do đó sẽ có 938 cập nhật mỗi lần lặp. Bằng cách xem xét nhiều dữ liệu hơn cho mỗi lần cập nhật trong mỗi lần lặp, nhiễu được tạo ra sẽ ít hơn so với stochastic gradient descent. Với ít nhiễu hơn so với phương pháp stochastic gradient descentn và nhiều cập nhật hơn trên mỗi lần lặp so với phương pháp batch gradient descent, mini-batch gradient descent có thể đạt được độ hội tụ nhanh nhất đáng kể.

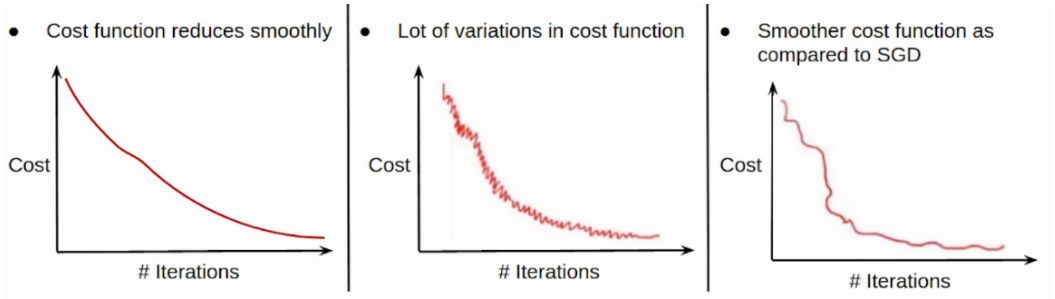
- Số lượng dữ liệu cho mỗi lần update
  - Trong batch gradient Descent, chúng ta lấy toàn bộ tập dữ liệu > tính hàm chi phí > cập nhật tham số.



Hình 1.6: Quỹ đạo của Batch , Mini Batch and Stochastic gradient descent

- Trong trường hợp Stochastic Gradient Descent, ta cập nhật các tham số sau mỗi lần quan sát và biết rằng khi các trọng số được cập nhật, nó được gọi là một lần lặp.
- Trong trường hợp Mini-batch Gradient Descent, ta chỉ lấy một tập hợp con dữ liệu và cập nhật các tham số dựa trên mỗi tập hợp con.

- Cost function



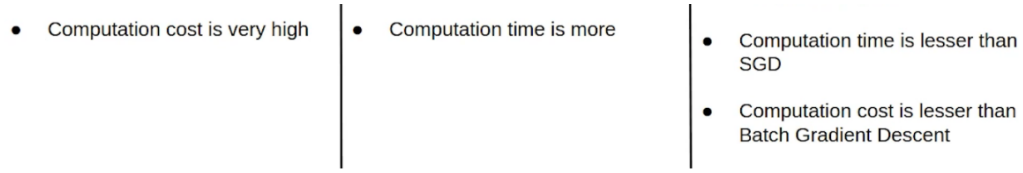
Hình 1.7: Hàm mất mát

- Cập nhật các tham số bằng cách sử dụng toàn bộ tập dữ liệu trong trường hợp Batch GD, nên hàm chi phí, trong trường hợp này, sẽ giảm đi một cách

trơn tru.

- Mặt khác, bản cập nhật này trong trường hợp của SGD không được suôn sẻ cho lắm. Vì ta đang cập nhật các tham số dựa trên một quan sát duy nhất nên có rất nhiều lần lặp lại. Cũng có thể mô hình cũng bắt đầu học tiếng ồn.
- Việc cập nhật hàm chi phí trong trường hợp Mini-batch Gradient Descent mượt mà hơn so với hàm chi phí trong SGD. Vì không cập nhật các tham số sau mỗi lần quan sát mà sau mỗi tập hợp con dữ liệu.

- Chi phí và thời gian tính toán



Hình 1.8: Chi phí và thời gian tính toán

- Vì phải tải toàn bộ tập dữ liệu cùng một lúc, thực hiện việc truyền tiến trên đó và tính toán lỗi, sau đó cập nhật các tham số, nên chi phí tính toán trong trường hợp Batch gradient descent là rất cao.
- Chi phí tính toán trong trường hợp SGD ít hơn so với Batch gradient descent vì chúng tôi phải tải từng quan sát tại một thời điểm nhưng thời gian tính toán ở đây sẽ tăng lên vì sẽ có nhiều cập nhật hơn, dẫn đến số lần lặp lại nhiều hơn .
- Trong trường hợp Mini-batch Gradient Descent, việc lấy một tập hợp con dữ liệu sẽ có số lần lặp hoặc cập nhật ít hơn và do đó thời gian tính toán trong trường hợp Mini-batch Gradient Descent sẽ nhỏ hơn SGD. Ngoài ra, vì không tải toàn bộ tập dữ liệu cùng một lúc trong khi tải một tập hợp con dữ liệu nên chi phí tính toán cũng thấp hơn so với phương pháp Batch gradient descent.

Chương 2

Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó

2.1 Tìm hiểu về Continual Learning khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó

Continual Learning tập trung vào việc cập nhật mô hình khi có dữ liệu mới, điều này làm cho mô hình theo kịp các phân phối dữ liệu hiện tại. Sau khi mô hình được cập nhật, nó không thể được phát hành một cách mù quáng vào sản xuất. Nó cần phải được thử nghiệm để đảm bảo rằng nó an toàn và tốt hơn so với mô hình hiện tại đang được sử dụng.
Continual Learning thường bị hiểu sai:

- Học liên tục KHÔNG ám chỉ một loại đặc biệt của thuật toán máy học cho phép cập nhật tăng dần của mô hình khi mỗi điểm dữ liệu mới xuất hiện. Ví dụ về loại đặc biệt này bao gồm việc cập nhật tuần tự Bayesian và các bộ phân loại KNN. Lớp thuật toán này khá nhỏ và đôi khi được gọi là "thuật toán học trực tuyến".
  - Khái niệm về học liên tục có thể được áp dụng cho bất kỳ thuật toán máy học giám sát nào, không chỉ là một lớp đặc biệt.
- Học liên tục KHÔNG có nghĩa là bắt đầu một công việc đào tạo lại mỗi khi một mẫu dữ liệu mới xuất hiện. Trên thực tế, điều này nguy hiểm vì khiến cho các mạng nơ-ron dễ bị quên mất nhanh chóng.
  - Hầu hết các công ty sử dụng học liên tục cập nhật mô hình của họ trong các micro-batch (ví dụ mỗi 512 hoặc 1024 ví dụ). Số lượng ví dụ tối ưu phụ thuộc vào công việc cụ thể.

Continual Learning có vẻ giống như một công việc của nhà khoa học dữ liệu ở mức độ bề mặt. Tuy nhiên, thường thì nó đòi hỏi rất nhiều công việc cơ sở hạ tầng để kích hoạt nó. Học liên tục đã được triển khai thành công trong công nghiệp. Tuy nhiên, nó đối mặt với một số thách thức lớn mà các công ty phải vượt qua.

2.1.1 Thách thức

- **Truy cập Dữ liệu Mới:**
  - Để cập nhật mô hình mỗi giờ, cần có dữ liệu huấn luyện được gắn nhãn chất lượng hàng giờ. Nhưng việc cập nhật nhanh chóng đặt ra thách thức nghiêm trọng.

- **Vấn đề:** Tốc độ lưu trữ dữ liệu vào kho dữ liệu.
    - Dữ liệu từ nhiều nguồn được đưa vào kho dữ liệu bằng cách sử dụng các cơ chế khác nhau và ở tốc độ khác nhau.
    - Có chiến lược lấy dữ liệu trực tiếp từ quá trình vận chuyển theo thời gian thực để huấn luyện trước khi đưa vào kho lưu trữ. Tuy nhiên, đối mặt với thách thức nhận diện và thu thập dữ liệu từ các nguồn không kiểm soát.
- **Vấn đề:** Tốc độ ghi nhãn.
    - Tốc độ gắn nhãn dữ liệu mới thường là điểm nghẽn.
    - Các ứng cử viên tốt cho học liên tục là những nhiệm vụ được gắn nhãn tự nhiên với vòng phản hồi ngắn.
    - Cần xem xét kỹ thuật giám sát yếu hoặc bán giám sát nếu không dễ dàng có được nhãn tự nhiên.
    - Tính toán nhãn có thể thực hiện theo đợt hoặc trực tiếp từ quá trình vận chuyển thời gian thực.

- **Đánh Giá:**
    - Học liên tục có thể dẫn đến thất bại của mô hình nếu cập nhật quá thường xuyên.
    - Có nguy cơ tấn công đối nghịch để đầu độc mô hình.
    - Việc kiểm tra đòi hỏi thời gian, giới hạn về tần suất cập

- **Chia Tỉ Lệ Dữ Liệu:**
    - Tính toán thường yêu cầu chia tỷ lệ. Chia tỷ lệ thường yêu cầu quyền truy cập vào số liệu thống kê dữ liệu toàn cầu như tối thiểu, tối đa, trung bình và phương sai.
    - Nếu sử dụng phương pháp đào tạo trạng thái, số liệu thống kê toàn cầu phải xem xét cả dữ liệu trước đó đã được sử dụng để huấn luyện mô hình cộng với dữ liệu mới đang được sử dụng để làm mới mô hình. Việc theo dõi số liệu thống kê toàn cầu trong trường hợp này có thể khó khăn.
    - Một kỹ thuật phổ biến để thực hiện việc này là tính toán hoặc ước tính các thống kê này tăng dần khi quan sát dữ liệu mới (ngược lại với việc tải toàn bộ tập dữ liệu vào thời gian đào tạo và tính toán từ đó).

- **Thuật toán**

– Thách thức nảy sinh khi sử dụng một số loại thuật toán cụ thể và muốn thực hiện cập nhật chúng một cách nhanh chóng (ví dụ: sau mỗi giờ).
– Những thuật toán này đòi hỏi quyền truy cập vào toàn bộ tập dữ liệu để được huấn luyện, ví dụ như các mô hình dựa trên ma trận, giảm kích thước và cây. Các mô hình này không thể được huấn luyện tăng dần bằng dữ liệu mới như mạng thần kinh hoặc các mô hình dựa trên trọng số khác có thể.
– Vấn đề xảy ra khi cần cập nhật chúng một cách nhanh chóng mà không thể đợi thuật toán xem xét toàn bộ tập dữ liệu.
– Một số biến thể của các mô hình đã được thiết kế để huấn luyện tăng dần, nhưng việc sử dụng chúng không phổ biến. Một ví dụ là Cây Hoeffding và các biến thể phụ.

2.1.2 Khái niệm Stateless retraining VS Stateful training

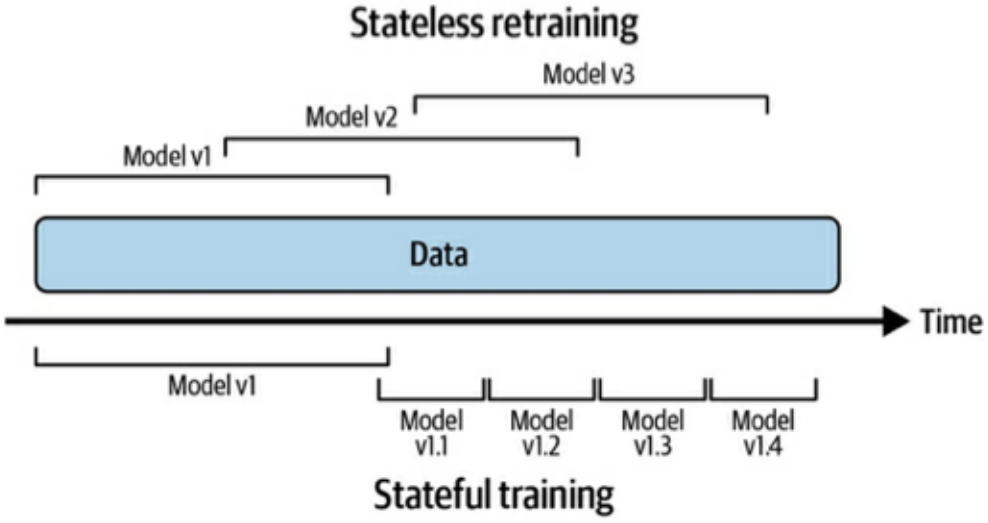


Figure 9-2. Stateless retraining versus stateful training

Hình 2.1: Stateless retraining VS Stateful training

1. Stateless retraining - Huấn luyện lại không trạng thái
   Huấn luyện lại mô hình từ đầu mỗi lần, sử dụng trọng số được khởi tạo ngẫu nhiên và dữ liệu mới hơn.

   - Có thể có một số trùng lặp với dữ liệu đã được sử dụng để huấn luyện phiên bản mô hình trước đó.
   - Hầu hết các công ty bắt đầu thực hiện việc học tập liên tục bằng cách sử dụng huấn luyện lại không trạng thái.

2. Stateful training - Huấn luyện có trạng thái
   Khởi tạo mô hình với các trọng số từ vòng huấn luyện trước và tiếp tục huấn luyện bằng cách sử dụng dữ liệu mới chưa từng thấy.

   - Cho phép mô hình cập nhật với lượng dữ liệu ít hơn đáng kể.
   - Cho phép mô hình hội tụ nhanh hơn và sử dụng ít năng lượng tính toán hơn.
   - Về mặt lý thuyết, nó có thể tránh việc lưu trữ dữ liệu hoàn toàn sau khi dữ liệu đã được sử dụng để huấn luyện (và để lại một khoảng thời gian an toàn). Điều này giúp loại bỏ những lo ngại về quyền riêng tư dữ liệu.
   - Thỉnh thoảng cần phải chạy huấn luyện lại không trạng thái với một lượng lớn dữ liệu để hiệu chỉnh lại mô hình.
   - Sau khi cơ sở hạ tầng được thiết lập chính xác, việc thay đổi từ huấn luyện lại không trạng thái sang huấn luyện có trạng thái sẽ trở thành một nút nhấn.
   - **Lặp lại mô hình so với lặp lại dữ liệu:** Huấn luyện trạng thái chủ yếu được sử dụng để kết hợp dữ liệu mới vào kiến trúc mô hình cố định và hiện có (tức là lặp lại dữ liệu). Nếu muốn thay đổi các tính năng hoặc kiến trúc của mô hình, bạn sẽ cần phải thực hiện quá trình huấn luyện lại không trạng thái lần đầu tiên.

2.1.3 Các giai đoạn của Continual Learning

Thường tiến hành Continual Learning theo bốn giai đoạn.

Giai đoạn 1: Huấn luyện lại thủ công, không lưu trạng thái

Mô hình chỉ được huấn luyện lại khi hai điều kiện được đáp ứng: (1) hiệu suất của mô hình đã giảm đến mức nó gây hại nhiều hơn là giúp ích, (2) đội ngũ của bạn có thời gian để cập nhật nó.

Giai đoạn 2: Huấn luyện lại tự động theo lịch trình cố định, không lưu trạng thái

- Giai đoạn này thường xảy ra khi các mô hình chính của một lĩnh vực đã được phát triển và do đó, ưu tiên của bạn không còn là tạo ra các mô hình mới, mà là duy trì và cải thiện những mô hình hiện tại. Việc ở giai đoạn 1 đã trở nên quá khó khăn để bỏ qua.
- Tần suất huấn luyện lại ở giai đoạn này thường dựa trên "ý kiến chủ quan".

- Điểm chuyển đổi giữa giai đoạn 1 và giai đoạn 2 thường là một đoạn mã mà ai đó viết để chạy việc huấn luyện lại không lưu trạng thái theo định kỳ. Việc viết mã này có thể dễ hoặc khó tùy thuộc vào số lượng phụ thuộc cần phối hợp để huấn luyện lại một mô hình.

- Các bước cấp cao của đoạn mã này là:

    1. Lấy dữ liệu.
    2. Giảm mẫu hoặc tăng mẫu nếu cần thiết.
    3. Trích xuất đặc trưng.
    4. Xử lý và/hoặc gán nhãn để tạo dữ liệu huấn luyện.
    5. Khởi đầu quá trình huấn luyện.
    6. Đánh giá mô hình mới.
    7. Triển khai nó.

- Có 2 phần cơ sở hạ tầng bạn sẽ cần triển khai đoạn mã:

    – Lập lịch
    – Kho mô hình để tự động phiên bản và lưu trữ tất cả các sản phẩm cần thiết để tái tạo mô hình. Các kho mô hình chín muối là AWS SageMaker và Databrick's MLFlow.

Giai đoạn 3: Huấn luyện tự động theo lịch trình, lưu trạng thái
Để đạt được điều này, bạn cần cấu hình lại đoạn mã của mình và một cách để theo dõi dòng dữ liệu và nguồn gốc của mô hình. Một ví dụ về một ví dụ đơn giản về phiên bản hóa dòng mô hình:

- V1 so với V2 là hai kiến trúc mô hình khác nhau cho cùng một vấn đề.

- V1.2 so với V2.3 có nghĩa là kiến trúc mô hình V1 đang ở lần lặp thứ 2 của quá trình huấn luyện không lưu trạng thái và V2 đang ở lần lặp thứ 3.

- V1.2.12 so với V2.3.43 có nghĩa là đã có 12 quá trình huấn luyện lưu trạng thái thực hiện trên V1.2 và 43 quá trình trên V2.3.

Có thể cần sử dụng điều này cùng với các kỹ thuật phiên bản khác như phiên bản dữ liệu để theo dõi toàn bộ hình ảnh về cách các mô hình đang phát triển. Theo tác giả, bà không biết có bất kỳ kho mô hình nào có khả năng theo dõi dòng mô hình như vậy nên các công ty xây dựng chúng trong nhà.

Ở bất kỳ thời điểm nào, bạn sẽ có nhiều mô hình đang chạy trong sản xuất cùng một lúc thông qua các sắp xếp được mô tả trong việc Kiểm thử mô hình trong Môi trường sản xuất. **Giai đoạn 4: Học liên tục**
Ở giai đoạn này, phần cố định theo lịch trình của các giai đoạn trước được thay thế bằng một cơ chế kích hoạt lại huấn luyện nào đó. Các kích hoạt này có thể là:

- Dựa trên thời gian.
- Dựa trên hiệu suất: ví dụ như hiệu suất đã giảm xuống dưới x%.
- Bạn không luôn có thể đo lường trực tiếp độ chính xác trong sản xuất, vì vậy bạn có thể cần sử dụng một số proxy yếu hơn.
- Dựa trên lượng: Tổng số dữ liệu được gán nhãn tăng lên 5%.
- Dựa trên sự thay đổi: ví dụ như khi phát hiện có sự chuyển động "lớn" trong phân phối dữ liệu.

Vấn đề khó khăn nhất khi sử dụng cơ chế kích hoạt dựa trên sự thay đổi là, như đã đề cập trong chương trước, sự chuyển động trong phân phối dữ liệu chỉ là vấn đề nếu nó làm suy giảm hiệu suất của mô hình của bạn. Việc xác định khi nào điều này xảy ra có thể là một vấn đề khó khăn.

2.1.4 Tần số cập nhật mô hình

Chúng ta cần hiểu và xác định mức lợi ích ta nhận được khi cập nhật mô hình của mình với dữ liệu mới. Muốn đạt được càng nhiều thì càng phải huấn luyện lại thường xuyên.
Đo lường độ mới của dữ liệu
Để định lượng giá trị của dữ liệu mới hơn là huấn luyện cùng một kiến trúc mô hình với dữ liệu từ 3 khoảng thời gian khác nhau, sau đó kiểm tra từng mô hình dựa trên dữ liệu được gắn nhãn hiện tại.

Nếu bạn phát hiện ra rằng việc để mô hình cũ trong 3 tháng sẽ gây ra sự khác biệt 10% về độ chính xác của dữ liệu thử nghiệm hiện tại và 10% là không thể chấp nhận được, thì bạn cần huấn luyện lại sau chưa đầy 3 tháng.
Lặp lại mô hình khi nào

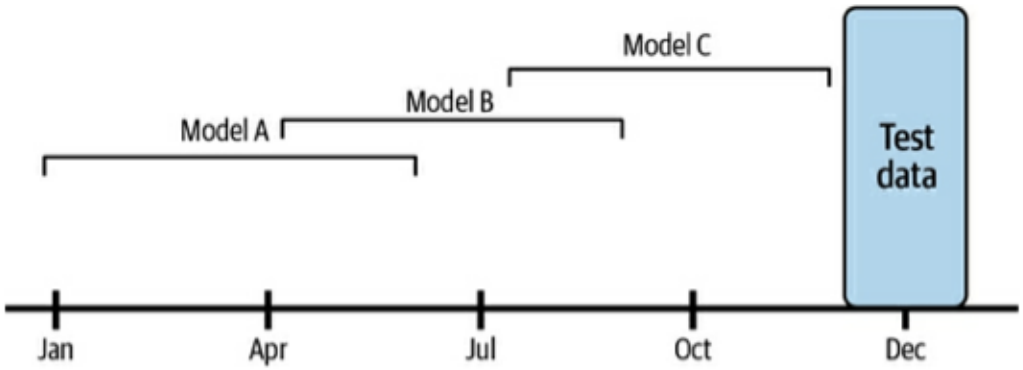


Figure 9-5. To get a sense of the performance gain you can get from fresher data, train your model on data from different time windows in the past and test on data from today to see how the performance changes

Hình 2.2: Ví dụ về multi-month datasets

- Nếu ta tiếp tục giảm kích hoạt đào tạo lại việc lặp lại dữ liệu và không thu được nhiều lợi ích, có lẽ bạn nên đầu tư vào việc tìm kiếm một mô hình tốt hơn (tất nhiên là nếu doanh nghiệp của bạn cần một mô hình).
- Nếu việc thay đổi sang kiến trúc mô hình lớn hơn yêu cầu sức mạnh tính toán 100X sẽ cải thiện hiệu suất 1%, nhưng việc giảm thời gian kích hoạt đào tạo lại xuống còn 3 giờ cũng giúp bạn tăng hiệu suất 1% với sức mạnh tính toán 1X, hãy ưu tiên việc lặp lại dữ liệu hơn việc lặp lại mô hình.

2.2 Tìm hiểu về Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó

Để kiểm tra đầy đủ các mô hình trước khi phổ biến rộng rãi, cần đánh giá ngoại tuyến trước khi triển khai và thử nghiệm trong sản xuất. Chỉ đánh giá ngoại tuyến là không đủ.
Lý tưởng nhất là mỗi nhóm đưa ra một quy trình rõ ràng về cách đánh giá các mô hình: thử nghiệm nào sẽ chạy, ai thực hiện chúng và các ngưỡng áp dụng để thúc đẩy mô hình lên giai đoạn tiếp theo. Tốt nhất là các quy trình đánh giá này được tự động hóa và khởi động khi có bản cập nhật mô hình mới. Việc thăng cấp giai đoạn cần được xem xét tương tự như cách đánh giá CI/CD trong công nghệ phần mềm.

2.2.1 Đánh giá ngoại tuyến trước khi triển khai

Hai cách phổ biến nhất là (1) Sử dụng phần tách thử nghiệm để so sánh với đường cơ sở và (2) chạy thử nghiệm ngược.

- **Test splits** thường ở dạng tĩnh để bạn có điểm chuẩn đáng tin cậy để so sánh nhiều mô hình. Điều này cũng có nghĩa là hiệu suất tốt trên phần phân chia thử nghiệm tĩnh cũ không đảm bảo hiệu suất tốt trong điều kiện phân phối dữ liệu hiện tại trong sản xuất.

- **Backtesting - Kiểm tra ngược** là ý tưởng sử dụng dữ liệu được gắn nhãn mới nhất mà mô hình chưa thấy trong quá trình huấn luyện để kiểm tra hiệu suất (ví dụ: nếu đã sử dụng dữ liệu của ngày cuối cùng, hãy sử dụng dữ liệu của giờ cuối cùng để kiểm tra lại).

2.2.2 Chiến lược Testing in Production Strategies

1. **Shadow Deployment**: Triển khai mô hình thách đấu song song với mô hình tốt nhất hiện có. Gửi mọi yêu cầu đến cả hai mô hình nhưng chỉ phục vụ suy luận của mô hình tốt nhất. Ghi lại các dự đoán cho cả hai mô hình để so sánh chúng.

    - **Ưu điểm**:
        - Đây là cách an toàn nhất để triển khai mô hình. Ngay cả khi mô hình mới có lỗi, dự đoán sẽ không được thực hiện.
        - Nó đơn giản về mặt khái niệm.

- Thử nghiệm sẽ thu thập đủ dữ liệu để đạt được ý nghĩa thống kê nhanh hơn tất cả các chiến lược khác vì tất cả các mô hình đều nhận được lưu lượng truy cập đầy đủ.

- **Nhược điểm**:
  - Không thể sử dụng kỹ thuật này để đo lường hiệu suất của mô hình phụ thuộc vào việc quan sát cách người dùng tương tác với các dự đoán. Ví dụ: các dự đoán từ mô hình đề xuất bóng tối sẽ không được cung cấp nên bạn sẽ không thể biết liệu người dùng có nhấp vào chúng hay không.
  - Kỹ thuật này tốn kém khi chạy vì nó tăng gấp đôi số lượng dự đoán và do đó số lượng tính toán cần thiết.

2. **A/B Testing**: Triển khai mô hình thách thức cùng với mô hình quán quân (mô hình A) và định tuyến phần trăm lưu lượng truy cập đến người thách thức (mô hình B). Dự đoán từ người thách đấu được hiển thị cho người dùng. Sử dụng tính năng giám sát và phân tích dự đoán trên cả hai mô hình để xác định xem thành tích của người thách đấu có tốt hơn về mặt thống kê so với nhà vô địch hay không.

   - **Ưu điểm**:
     - Đây là cách an toàn nhất để triển khai mô hình. Ngay cả khi mô hình mới có lỗi, dự đoán sẽ không được thực hiện.
     - Testing A/B rất dễ hiểu và có rất nhiều thư viện cũng như tài liệu hỗ trợ.
     - Không cần phải xem xét các trường hợp đặc biệt phát sinh từ các yêu cầu suy luận song song cho các chế độ dự đoán trực tuyến (xem nhược điểm của việc triển khai bóng *(shadow deploy)*).
   - **Nhược điểm**:
     - Nó kém an toàn hơn so với Shadow Deployment.
     - Quyền lựa chọn cố hữu giữa việc giả định nhiều rủi ro hơn (định tuyến nhiều lưu lượng truy cập hơn đến mô hình B) và lấy đủ mẫu để phân tích nhanh hơn.

Tài liệu tham khảo

[1] Funda, K. E. [2020]. Machine Learning cơ bản.