TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

KHOA CÔNG NGHỆ THÔNG TIN

NHẬP MÔN HỌC MÁY

Người hướng dẫn : **Thầy Lê Anh Cường**

Người thực hiện : **Bùi Văn Thống – 52100934**

Lớp : 21050301

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Đầu tiên, em xin phép gửi lời các thầy cô khoa Công Nghệ Thông Tin của trường Đại học Tôn Đức Thắng nói chung và đặc biệt là thầy Lê Anh Cường nói riêng. Chúng em xin cảm ơn thầy vì trong suốt thời gian học tập vừa qua thầy đã truyền tải cho chúng em nhưng kiến thức về môn học Nhập Môn Học Máy cũng như là các kiến thức để giúp chúng em hoàn thành được bài báo cáo này. Có thể trong quá trình làm bài cáo này sẽ phát sinh ra những sự cố không mong muốn, mong thầy có thể góp ý để chúng em có thể hoàn thành bài báo cáo một cách tốt nhất và tránh mắc phải những sự cố tương tự ở các môn học tiếp theo.

Chúng em cảm ơn thầy rất nhiều ạ!

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi / chúng tôi và được sự hướng dẫn của Giảng viên Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. 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.

Nếu phát hiện có bất kỳ sự gian lận nào chúng tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do chúng tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 21 tháng 12 năm 2023
Tác giả
(ký tên và ghi rõ họ tên)

Bùi Văn Thống

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

MỤC LỤC

MỤC LỤC BẢNG

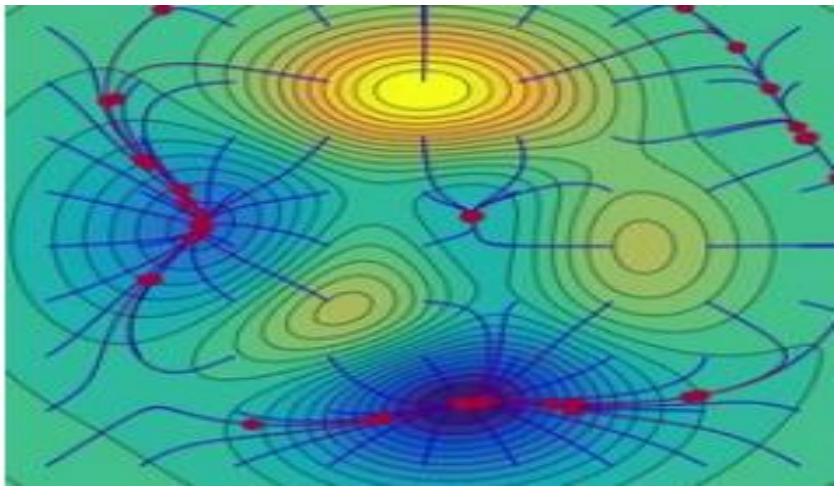
1 Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình

1.1 Các phương pháp Optimizer trong huấn luyện mô hình:

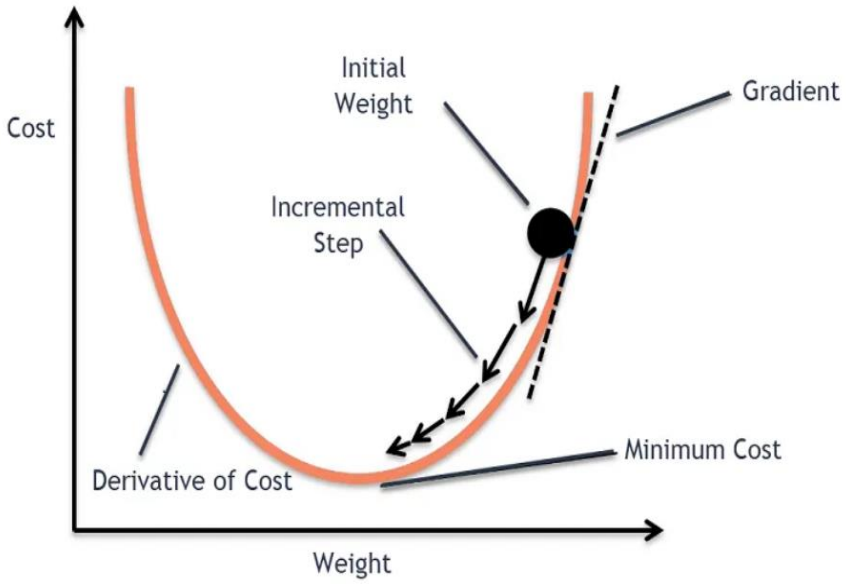
1.1.1 Gradient Descent (GD):

1.1.1.1 Giới thiệu về Gradient Descent

- Gradient Descent là một trong các thuật toán tối ưu hóa phổ biến nhất, đặc biệt là trong lĩnh vực Machine Learning. Thuật toán này dựa trên việc tìm kiếm cực trị (cực đại hoặc cực tiểu) của một hàm số bằng cách tính đạo hàm và di chuyển theo hướng giảm dần của gradient.



Hình 1. 1 Gradient Descent



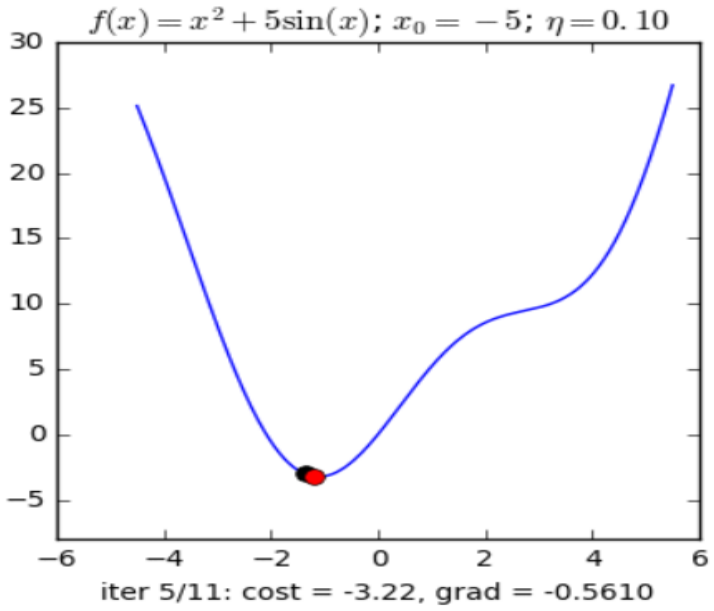
Hình 1. 2 Gradient Descent

1.1.1.2 Hàm một biến

- Xét hàm một biến $f(x)$. Thuật toán Gradient Descent tìm cực tiểu của hàm số bằng cách khởi tạo giá trị x tại một vị trí ngẫu nhiên, sau đó di chuyển x ngược hướng với đạo hàm của $f(x)$. Thao tác này sẽ được lặp lại cho đến khi đạt đến một ngưỡng nào đó.
- Công thức cập nhật x trong Gradient Descent:

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

- Trong đó:
  - $x_t$là giá trị x tại bước thứ t
  - α là learning rate (tốc độ học)
  - $'_{x_t}$ là đạo hàm của hàm f tại x_t.



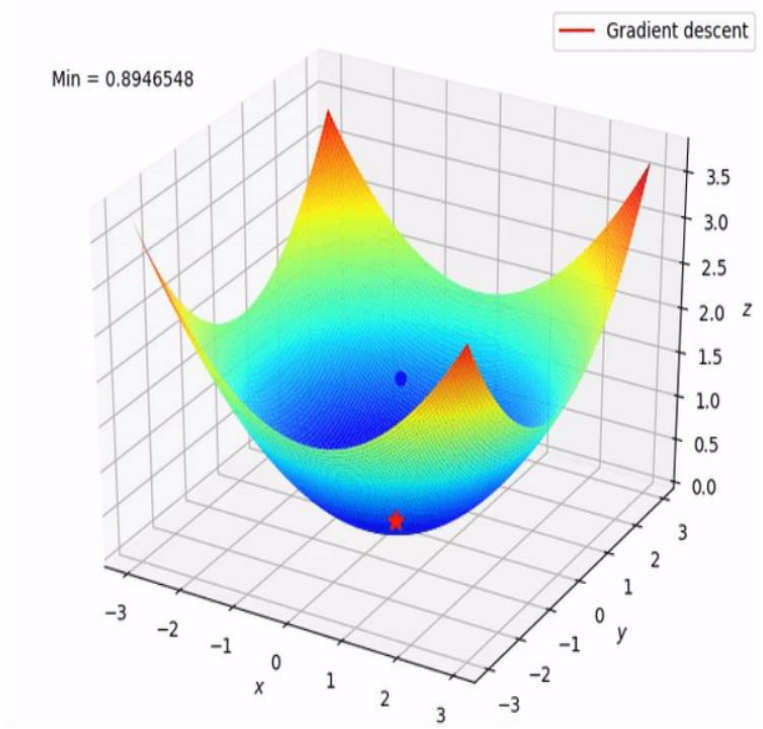
Hình 1. 3 Hàm một biến

1.1.1.3 Hàm nhiều biến

- Đối với một hàm nhiều biến $f(\vec{x})$, thuật toán này sẽ tính Gradient (vector đạo hàm) của hàm f tại một điểm ngẫu nhiên $\vec{x}$ sau đó di chuyển $\vec{x}$ ngược hướng với Gradient này.
- Công thức cập nhật \vec{x}:

$$\vec{x}_{t+1} = \vec{x}_t - \alpha * \nabla f(\vec{x}_t)$$

- Trong đó:
  - $\vec{x}_t$ là giá trị $\vec{x}$ tại bước thứ t
  - α là learning rate (tốc độ học)
  - $\nabla f(\vec{x})$ là Gradient của hàm f tại $\vec{x}_t$.



Hình 1. 4 Hàm nhiều biến

1.1.1.4 Các thuật toán Gradient Descent và Biến thể

– Có nhiều biến thể của thuật toán này tuỳ thuộc vào việc lựa chọn dữ liệu huấn luyện:

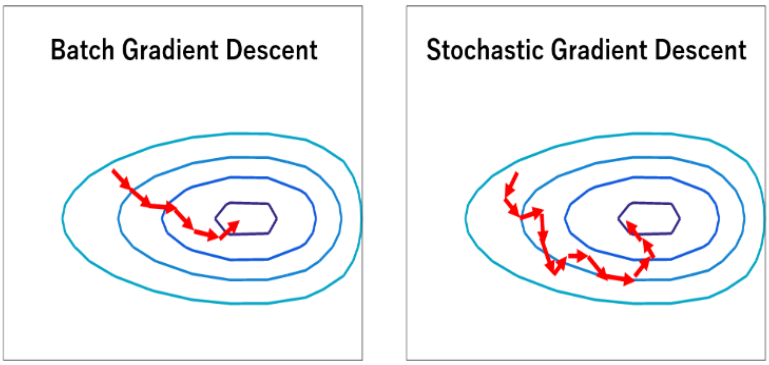
- Batch Gradient Descent: Sử dụng toàn bộ dữ liệu huấn luyện để cập nhật $\vec{x}$ trong mỗi bước lặp. Thuật toán này có độ chính xác cao nhưng mất nhiều thời gian do tính toán trên toàn bộ dữ liệu.
- Stochastic Gradient Descent (SGD): Sử dụng chỉ một điểm dữ liệu huấn luyện ngẫu nhiên để cập nhật $\vec{x}$. Tốc độ hội tụ nhanh hơn, nhưng độ chính xác thấp hơn so với Batch Gradient Descent.

$$\theta = \theta - \eta \nabla_\theta J(\theta; \mathbf{x}_i; \mathbf{y}_i)$$

- Mini-batch Gradient Descent: Sử dụng một số lượng nhỏ điểm dữ liệu (1 < k < n) huấn luyện (mini-batch) để cập nhật $\vec{x}$. Kết hợp ưu điểm của cả Batch Gradient Descent và SGD.

$$\theta = \theta - \eta \nabla_\theta J(\theta; \mathbf{x}_{i:i+n}; \mathbf{y}_{i:i+n})$$

- So sánh:
  - Vì SGD sử dụng từng mẫu đơn cho tốc độ tính toán đạo hàm nhanh hơn nhưng tốc độ hội tụ của nó lâu hơn so với Mini-batch GD.



Hình 1. 5 So sánh Batch vs Stochastic Gradient Descent

  - Mini-batch GD thường được sử dụng hơn vì:
    - Tốc độ hội tụ nhanh hơn Stochastic Gradient Descent.
    - Tốc độ tính đoạ hàm nhanh hơn Batch Gradient Descent.

– Biến thể cải tiến của Gradient Descent:

- Momentum: Giảm dao động qua lại của gradient và đi nhanh hơn dọc theo hướng tiến.
- Nesterov Accelerated Gradient (NAG): Sử dụng momentum bằng cách tính gradient trước khi cập nhật vị trí của $\vec{x}$.
- Adaptive Gradient (Adagrad): Đưa vào learning rate riêng cho mỗi parameter.
- Adaptive Moment Estimation (Adam): Kết hợp momentum và adaptive learning rate.

1.1.1.5 Điều kiện dừng của Gradient Descent

– Số lượng vòng lặp cố định: Dừng sau một số lượng lần cập nhật $\vec{x}$ cố định.

- Đạo hàm tiệm cận 0: Dừng khi đạo hàm của hàm f tiệm cận 0 ở mỗi biến.
- Thay đổi của hàm tiệm cận 0: Dừng khi thay đổi của hàm f qua mỗi bước lặp tiệm cận 0.

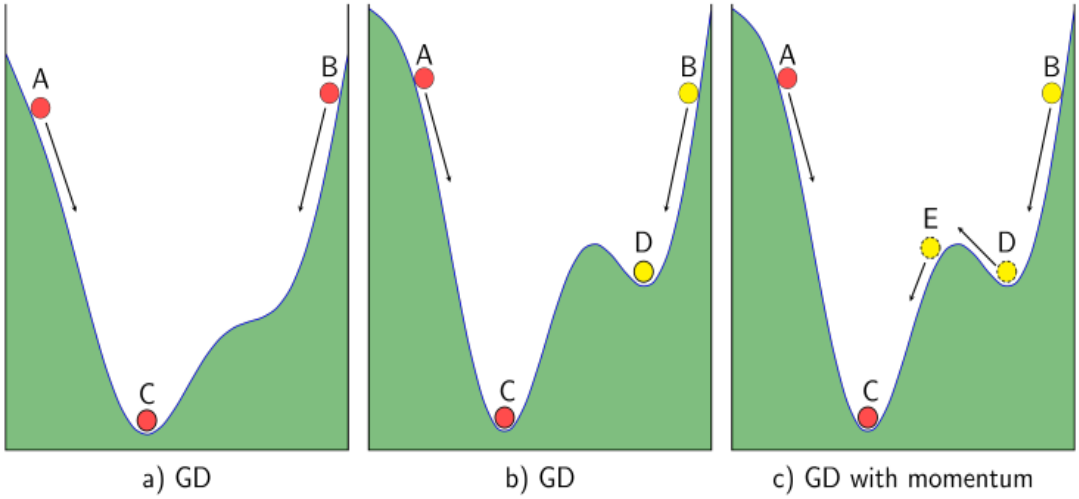
1.1.1.6 Ưu, nhược điểm

- Ưu điểm:
    - Thuật toán gradient descent cơ bản, dễ hiểu. Thuật toán đã giải quyết được vấn đề tối ưu model neural network bằng cách cập nhật trọng số sau mỗi vòng lặp.
- Nhược điểm:
    - Vì đơn giản nên thuật toán Gradient Descent còn nhiều hạn chế như phụ thuộc vào nghiệm khởi tạo ban đầu và learning rate.
    - Ví dụ 1 hàm số có 2 global minimum thì tùy thuộc vào 2 điểm khởi tạo ban đầu sẽ cho ra 2 nghiệm cuối cùng khác nhau.
    - Tốc độ học quá lớn sẽ khiến cho thuật toán không hội tụ, quanh quẩn bên đích vì bước nhảy quá lớn; hoặc tốc độ học nhỏ ảnh hưởng đến tốc độ training.

1.1.2 Momentum

- Thuật toán GD thường được ví với tác dụng của trọng lực lên một hòn bi đặt trên một mặt có dạng như hình một thung lũng như hình a. Bất kể ta đặt hòn bi ở A hay B thì cuối cùng sẽ lăn xuống vị trí kết thúc C. Tuy nhiên, nếu như bề mặt có hai đấy thung lũng như hình b thì tuỳ vào việc đặt bi ở A hay B, vị trí cuối cùng của bi sẽ ở C hoặc D. Điểm D được gọi là một điểm cực tiểu địa phương chúng ta không mong muốn, vấn đề này chúng ta đã thảo luận ở phần các vấn đề của gradient descent bên trên.
- Nếu suy nghĩ một cách vật lý hơn, vẫn trong hình b, nếu vận tốc ban đầu của bi khi ở điểm B đủ lớn, khi bi lăn đến điểm D, theo đà, bi có thể tiếp tục di chuyển lên dốc phía bên trái của D. Và nếu giả sử vận tốc ban đầu lớn hơn nữa, bi có thể vượt dốc tới điểm E rồi lăn xuống C như trong hình c. Đây chính là điều chúng ta

mong muốn. Dựa trên hiện tượng này, một thuật toán được ra đời nhằm khắc phục việc nghiệm của GD rơi vào một điểm cực tiểu đại phương không mong muốn. Thuật toán đó có tên là Momentum



Hình 1. 6 Monmentum

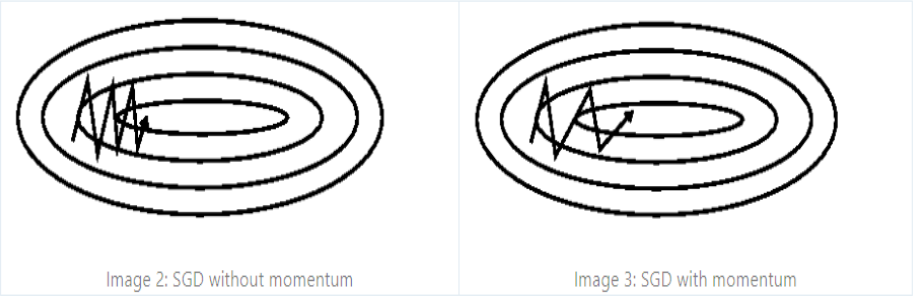
– Trong GD, chúng ta cần tính lượng thay đổi ở thời điểm t để cập nhật vị trí mới cho nghiệm (tức hòn bi). Nếu chúng ta coi đại lượng này như vận tốc v_t trong vậy lý, vị trí mới của hòn bi sẽ là $\theta_{t+1} = \theta_t - v_t$. Dấu trừ thể hiện việc phải di chuyển ngược với đạo hàm. Công việc của chúng 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 γ thường chọn là một giá trị khoảng 0.9, v_t-1 chính là vận tốc tại thời điểm trước đó, ∇θ J(θ) chính là gradient (độ dốc) của điểm trước đó. Sau đó vị trí mới của hòn bi được xác định như sau:

$$\theta = \theta - v_t$$

– Về cơ bản, khi sử dụng momentum giống như chúng ta đẩy một quả bóng xuống thung lũng. Qủa bóng tích luỹ vận tốc khi n lăn xuống và ngày càng nhanh hơn trên đường đi, và quả bóng này chịu tác dụng của lực cản như là không khí, chính vì thế mà chúng ta có tham số $\gamma < 1$



Hình 1. 7 SGD Momentum

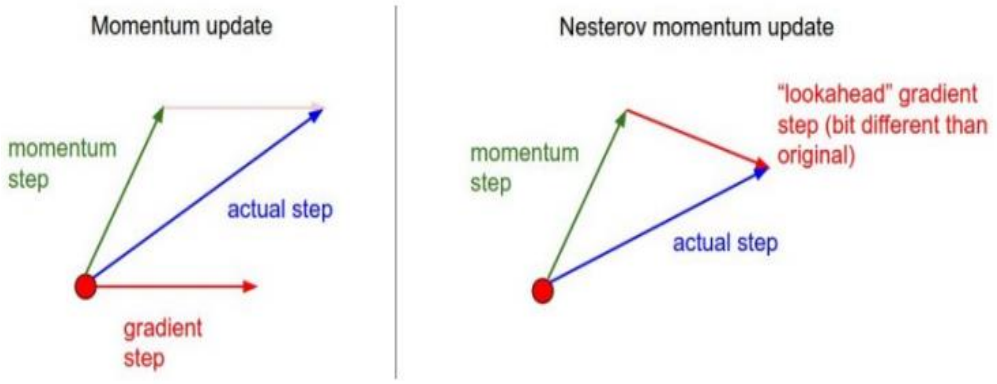
– Trong phần đầu tiên (Hình 2), SGD không có momentum cho thấy các đường đi dao động, cho thấy tính không hiệu quả và hội tụ chậm đến điểm tối thiểu. Ngược lại, Hình 3 miêu tả SGD với momentum có một đường đi mượt mà và nhanh hơn đến điểm tối thiểu, cho thấy tính hiệu quả của việc sử dụng momentum trong việc tăng tốc hội tụ.

1.1.3 Nesterow accelerated gradient

– Momentum giúp hòn bi vượt dốc cực tiểu địa phương, tuy nhiên, có một hạn chế là khi gần tới đích, momentum vẫn mất nhiều thời gian trước khi dừng lại. Điều này chính là do có đà. Nesterow accelerated gradient (viết tắt là NAG) giúp khắc phục điều này để cho thuật toán hội tụ nhanh hơn.

– Ý tưởng cơ bản là dự đoán hướng đi trong tương lai, tức nhìn trước một bước. Cụ thể, nếu sử dụng số hạng momentum γv_{t-1} để cập nhật thì ta có thể xấp xỉ được vị trí tiếp theo của hòn bi là $\theta - \gamma v_{t-1}-1$ (chúng ta không đính kèm phần gradient ở đây vì sẽ sử dụng ở bước cuối cùng). Vậy, thay vì sử dụng gradient của điểm hiện tại, NAG đi dự đoán trước bước tiếp theo.

– Theo dõi hình 1.3:



Hình 1.3: Ý tưởng của Nesterov accelerated gradient

Hình 1. 8 Nesterov accelerated gradient

- Với momentum thông thường: lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm hiện tại.
- Với Nesterove momentum: lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm được xấp xỉ là điểm tiếp theo.

– Từ trên, ta có công thức cập nhật của NAG như sau:

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

$$\theta = \theta - v_t$$

1.1.4 Adagrad

– Như đã thảo luận ở mục các vấn đề của gradient descent, thuật toán gradient descent áp dụng 1 learning rate cho tất cả các tham số. Tuy nhiên, trên thực tế, đôi khi có những tham số ít xuất hiện và cần cập nhật bước sải lớn hơn (high learning rate), các tham số xuất hiện thường xuyên cần cập nhật bước sải nhỏ hơn (low learning rate). Điều này thường xuyên gặp ở nhiều bài toán mà dữ liệu có dạng thưa (sparse). Adagrad ra đời nhằm giải quyết vấn đề này, nó điều chỉnh learning rate theo các tham số. Pennington và đồng nghiệp đã sử dụng Adagrad để train Glove (một mô hình word embedding trong NLP), vì những từ không xuất hiện

thường xuyên đòi hỏi cập nhật bước sải lớn hơn rất nhiều so với những từ thường xuyên và đã đạt được hiệu quả tốt hơn so với gradient descent.

- Trước đây, chúng ta thực hiện một update cho tất cả các tham số θ, cùng một lúc mỗi tham số θ_i sử dụng learning rate giống nhau η. Adagrad sử dụng learning rate khác nhau cho mỗi tham số θi tại mỗi bước t.
- Gọi gt là gradient tại bước t, $g_{t,i}$ là đạo hàm riêng của hàm mục tiêu theo tham số θ_i tại bước thứ t:

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

- Khi đó, GD update cho mỗi tham số θ_i tại bước thứ t trở thành:

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

- Trong quy tắc cập nhật của mình, Adagrad sửa đổi sinh learning rate cho mỗi bước t cho mỗi tham số θ_i dựa trên các gradient quá khứ đã được tính toán cho θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} . g_{t,i}$$

- Trong đó, $G_t \in R^{dxd}$ là một ma trận đường chéo mà các phần tử trên đường chéo i, i là tổng bình phương của các gradient theo θ_i cho đến bước thứ t, và ε là hệ số để tránh chia cho 0 (thường là 10^{-8}). Theo thử nghiệm, nếu không có căn bậc 2 dưới mẫu, thuật toán lại hoạt động tệ hơn nhiều. Như vậy, G_t gồm tổng bình phương các gradient quá khứ của tất cả các tham số θ dọc theo đường chéo, chúng ta có thể viết tổng quát lại công thức cập nhật cho Adagrad như sau:

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

  - Trong đó:
    - η: hằng số

- gt: gradient tại thời điểm t
- ϵ: hệ số tránh lỗi (chia cho mẫu bằng 0)
- G : là ma trận chéo mà mỗi phần tử trên đường chéo (i,i) là bình phương của đạo hàm vectơ tham số tại thời điểm t.

– Thông thường, η = 0.01. Rõ ràng, lợi ích của Adagrad là ước lượng để điều chỉnh learning rate cho từng tham số. Tuy nhiên, điểm yếu chính của Adagrad là sự tích luỹ gradient ở mẫu số lớn dần lên trong quá trình training. Điều này làm cho learning rate thu hẹp lại và dần trở lên vô cùng nhỏ, tại thời điểm đó, thuật toán không còn có thể học được nữa. Các thuật toán sau này nhằm giải quyết lỗ hổng này.

– Ưu điểm:

- Một lợi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, chỉ cần để tốc độ học default là 0.01 thì thuật toán sẽ tự động điều chỉnh.

– Nhược điểm:

- Yếu điểm của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm tốc độ học cực kì nhỏ, làm việc training trở nên đóng băng.

1.1.5 RMSprop

– RMSProp là phương pháp điều chỉnh learning rate được đưa ra bởi Geoff Hinton. RMSProp và Adadelta phát triển độc lập cùng lúc xuất phát từ nhu cầu giải quyết triệt để learning rate giảm dần của Adagrad bằng cách chia tỷ lệ học cho trung bình của bình phương gradient. RMSProp có công thức cập nhật như sau:

$$E[g^2]_t = 0.9[g^2]_{t-1} + 0.1_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

- RMSProp cũng chia learning rate cho $E[g^2]_t$. Hinton đề nghị $\gamma = 0.9$ và giá trị mặc định tốt cho $\eta = 0.001$.
- Ưu điểm:
  - Ưu điểm rõ nhất của RMSprop là giải quyết được vấn đề tốc độ học giảm dần của Adagrad (vấn đề tốc độ học giảm dần theo thời gian sẽ khiến việc training chậm dần, có thể dẫn tới bị đóng băng)
- Nhược điểm:
  - Thuật toán RMSprop có thể cho kết quả nghiệm chỉ là local minimum chứ không đạt được global minimum như Momentum. Vì vậy người ta sẽ kết hợp cả 2 thuật toán Momentum với RMSprop cho ra 1 thuật toán tối ưu Adam. Chúng ta sẽ trình bày nó trong phần sau.

1.1.6 Adam

- Adam, (Adaptive moment Estimation), là một phương pháp ước lượng learning rate cho mỗi tham số. Adam được coi như là sự kết hợp của AdaDelta hay RMSProp và momentum. Trong khi momentum có thể xem như một quả bóng đang chạy xuống dốc, Adam lại giống như một quả bóng nặng với ma sát. Adam sử dụng bình phương gradient để chia tỷ lệ learning rate như RMSProp và tận dụng "đà" giống như momentum. Công thức cập nhật đầy đủ của Adam:

$$\begin{cases} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon} \end{cases}$$

- Để tốt hơn về chi phí tính toán, ta có thể viết lại như sau:

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

$$\theta_t = \theta_{t-1} - \eta_t \frac{m_t}{\sqrt{v_t} + \varepsilon}$$

– Bias Correction

- Adam tính toán moment cấp 1 và moment cấp 2 của gradient để ước lượng learning rate cho các tham số.
- Để ước lượng các moment, Adam tận dụng exponentially moving average của gradient và bình phương gradient, với decay rate là β1 và β2.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

- Với m_t và v_t được gọi là moving averages, g là gradient trên mini-batch hiện tại. Để xem các giá trị này tương quan với moment như thế nào, chúng ta hãy xem xét giá trị kỳ vọng của các moving averages. Vì m và v là các ước lượng của moment cấp 1 và moment cấp 2, chúng ta muốn có tính chất sau:

$$\mathbb{E}[m_t] = \mathbb{E}[g_t]$$
$$\mathbb{E}[v_t] = \mathbb{E}[g_t^2]$$

- Ta khai triển v_t như sau (tương tự với m_t):

$$v_0 = 0$$
$$v_1 = \beta_2 v_0 + (1-\beta_2) g_1^2 = (1-\beta_2) g_1^2$$
$$v_2 = \beta_2 v_1 + (1-\beta_2) g_2^2 = \beta_2 (1-\beta_2) g_1^2 + (1-\beta_2) g_2^2$$
$$\dots$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 = (1-\beta_2) \sum_{i=1}^{t} \beta_2^{t-i} . g_i^2$$

- Tiếp theo, chúng ta cần điều chỉnh giá trị ước lượng của v_t để cho kỳ vọng của nó xấp xỉ moment cấp 2, sở dĩ có sự sai lệch đại lượng $(1 - \beta_2^t)$ là do ta đã khởi tạo v_0 = 0. Từ đó ta có công thức hiệu chỉnh cho v như sau:

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

- Tương tự, thực hiện các bước trên với m_t. Cuối cùng, chúng ta có công thức cập nhật tham số cho Adam như sau:

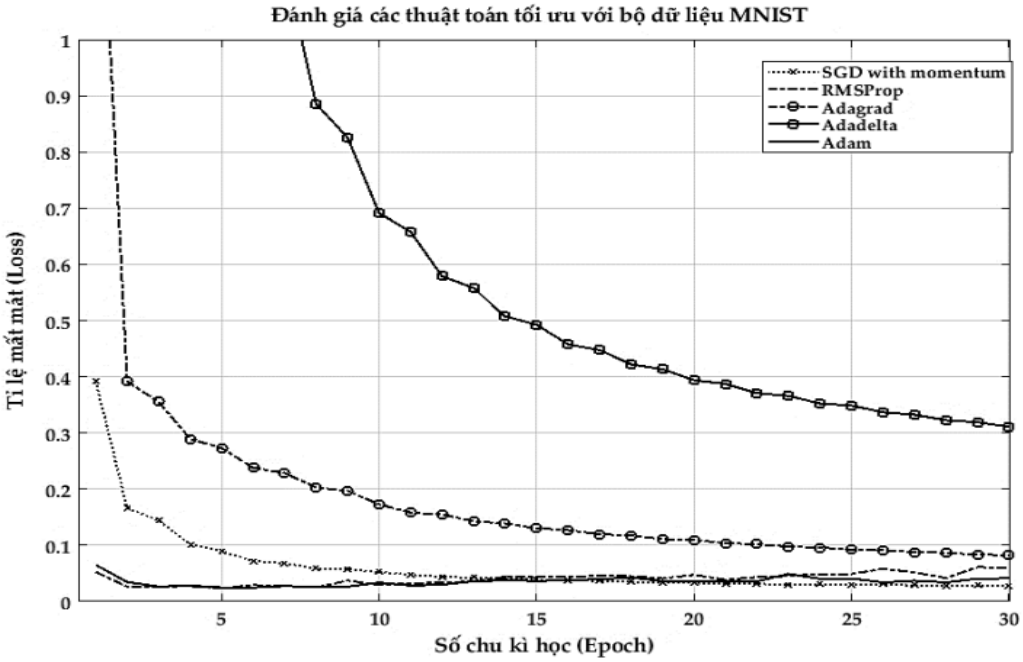
$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

1.2 Đánh giá kết quả của các thuật toán

1.2.1 Dữ liệu MNIST

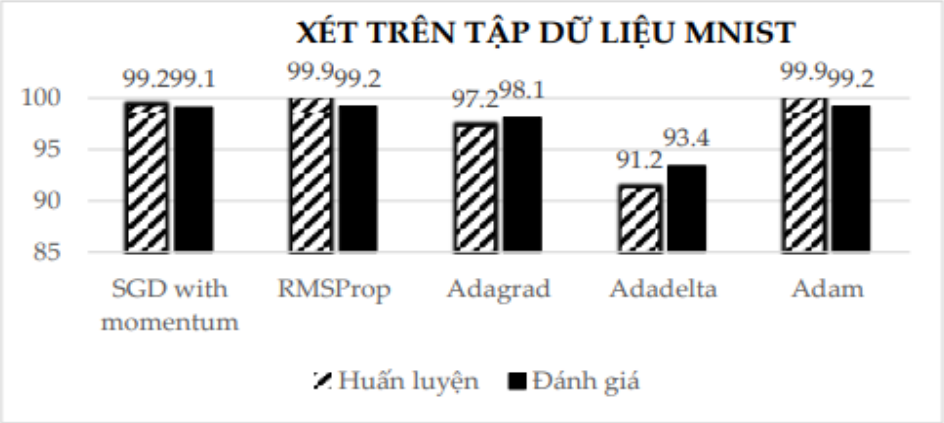
– Bộ dữ liệu MNIST là bộ dữ liệu gồm các hình ảnh xám (grayscale picture) các chữ số viết tay được chia sẻ bởi Yann Lecun bao gồm 70000 ảnh chữ số viết tay được chia thành 2 tập: tập huấn luyện gồm 60000 ảnh và tập kiểm tra 10000 ảnh. Các chữ số viết tay ở tập MNIST được chia thành 10 nhóm tương ứng với các chữ số từ 0 đến 9. Tất cả hình ảnh trong tập MNIST đều được chuẩn hóa với kích thước 28 x 28 điểm ảnh. Dưới đây là một số hình ảnh được trích xuất từ bộ dữ liệu

1.2.2 Kết quả với bộ cơ sở dữ liệu MNIST



Hình 1. 9 Tỉ lệ mất mát của các thuật toán tối ưu trên tập dữ liệu MNIST

- Hình 1.9 mô tả kết quả của loss function của các thuật toán, chỉ xét ở 30 chu kì học đầu để có cách nhìn cụ thể hơn về sự biến thiên của hàm mất mát.
- Từ đồ thị, có thể nhận thấy rằng, Adam và RMSProp là 2 thuật toán có biên độ dao động thấp nhất, gần như không thay đổi quá nhiều quanh giá trị 0.5. Trong khi đó, AdaDelta và AdaGrad là hai thuật toán có sự biến động lớn nhất trong suốt các chu kì học. Bên cạnh đó, nhận thấy rằng thuật toán SGD với động lượng là thuật toán có kết quả hội tụ nhanh nhất và tốt nhất là với tỉ lệ mất mát rơi vào khoảng 0.023. Các thuật toán Adam, RMSProp và Adagrad cũng có kết quả rất tốt lần lượt là 0.06, 0.067, 0.059, thuật toán Adelta có kết quả cao nhất trong các thuật toán đang xét với tỉ lệ mất mát 0.229.



Hình 1. 10 Tỉ lệ nhận dạng tập huấn luyện và tập đánh giá

– Từ hình 1.10, có thể thấy rằng tỉ lệ nhận dạng đúng của mô hình chịu sự ảnh hưởng từ các thuật toán tối ưu. Cụ thể, đối với thuật toán cho tỉ lệ mất mát cao như Adadelta hay Adagrad, tỉ lệ nhận dạng đúng khá thấp, rơi vào khoảng 93.4% trên tập đánh giá (Adadelta). Trong khi đó, các thuật toán cho tỉ lệ mất mát thấp như SGD with momentum, RMSProp và Adam cho tỉ lệ nhận dạng đúng khả quan hơn, đạt khoảng 99.2%, khi sử dụng trên cùng một mô hình kiến trúc mạng đề ra.

2 Tìm hiểu về Continual Learning và Test Production:

2.1 Continual Learning:

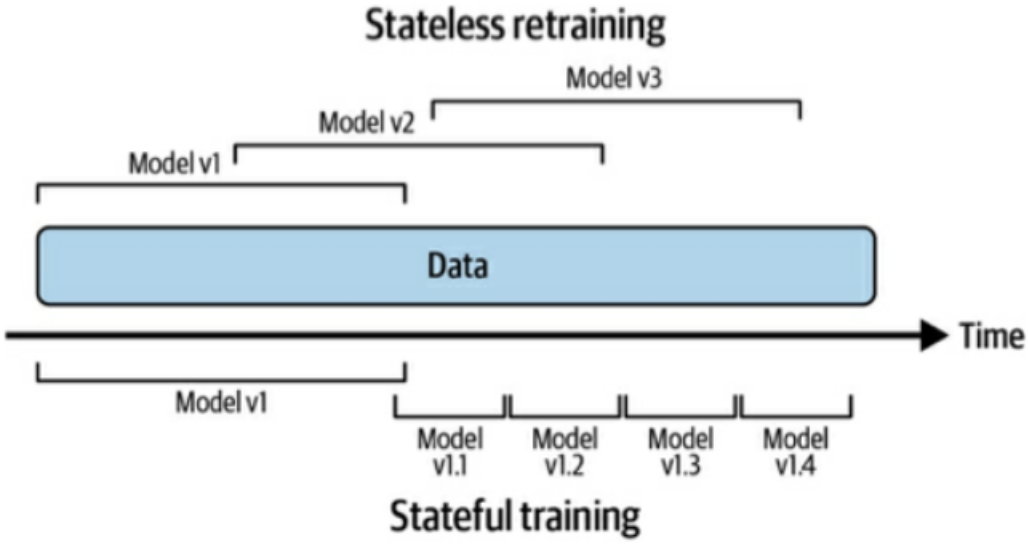
2.1.1 Giới thiệu

- Continual Learning là ý tưởng cập nhật mô hình của bạn khi có dữ liệu mới; điều này làm cho mô hình của bạn theo kịp các phân phối dữ liệu hiện tại.
- Continual Learning gọi là học suốt đời. Học máy suốt đời (Lifelong machine learning - LML) (hay đơn giản là học suốt đời) nhằm bắt chước quá trình và khả năng học của con người, tích luỹ và duy trì tri thức đã học được từ các bài toán trước và không ngừng sử dụng tri thức đó để học và giải quyết bài toán mới.

2.1.2 Continual Learning vì sao tốt

- Lý do cơ bản là giúp mô hình của bạn theo kịp sự thay đổi phân phối dữ liệu. Có một số trường hợp sử dụng trong đó việc thích ứng nhanh chóng với việc thay đổi phân phối là rất quan trọng. Ví dụ:
    - Trường hợp có thể xảy ra những thay đổi nhanh chóng và bất ngờ
    - Trường hợp không thể lấy dữ liệu huấn luyện cho một sự kiện cụ thể
    - Trường hợp sử dụng nhạy cảm đối với vấn đề khởi động lạnh (Vấn đề này xảy ra khi mô hình của bạn phải đưa ra dự đoán cho một người dùng mới (hoặc đã đăng xuất) mà không có dữ liệu lịch sử (hoặc dữ liệu đã lỗi thời). Nếu bạn không điều chỉnh mô hình của mình ngay khi bạn nhận được một số dữ liệu từ người dùng đó, bạn sẽ không thể đề xuất những thứ liên quan đến người dùng đó.)

2.1.3 Stateless retraining và Stateful training



Hình 2. 1 Stateless retraining vs Staeful training

2.1.3.1 Stateless retraining

- Stateless retraining là một phương pháp huấn luyện mô hình truyền thống, trong đó chúng ta đầu tiên huấn luyện một mô hình ban đầu trên tập huấn luyện gốc và sau đó huấn luyện lại nó khi có dữ liệu mới. Nói cách khác, mỗi lần bạn muốn cập nhật mô hình của mình, bạn sẽ phải huấn luyện lại từ đầu với dữ liệu mới nhất.
    - 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 Stateless retraining.

2.1.3.2 Stateful training

- Stateful training (còn được gọi là “fine-tuning” hoặc “incremental learning”) là một phương pháp huấn luyện mô hình trong đó chúng ta khởi tạo mô hình của mình với trọng số từ vòng luyện tập trước đó và tiếp tục luyện tập bằng cách sử dụng dữ liệu mới chưa được nhìn thấy.
- Phương pháp này cho phép mô hình của bạn cập nhật với lượng dữ liệu ít hơn đáng kể, hội tụ nhanh hơn và sử dụng ít năng lượng tính toán hơn. Một số công ty

đã báo cáo giảm đến 45% năng lượng tính toán. Lý thuyết cho thấy nó có thể loại bỏ 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 lý thuyết loại bỏ các vấn đề về quyền riêng tư dữ liệu. Thỉnh thoảng, bạn sẽ cần chạy lại huấn luyện không có trạng thái với một lượng lớn dữ liệu để hiệu chỉnh lại mô hình. Một khi cơ sở hạ tầng của bạn được thiết lập đúng cách, việc chuyển từ stateless retraining sang stateful training trở nên dễ dàng.

2.1.4 Thách thức Continual Learning

2.1.4.1 Truy cập dữ liệu mới

- Để cập nhật mô hình hàng giờ, bạn cần dữ liệu huấn luyện chất lượng được gán nhãn mỗi giờ. Tần suất cập nhật càng ngắn, thách thức càng trở nên quan trọng.
- Vấn đề tốc độ nạp dữ liệu vào kho dữ liệu
    - Khó khăn trong việc nạp dữ liệu vào kho từ các nguồn khác nhau và tốc độ khác nhau.
    - Phương pháp phổ biến: rút dữ liệu trực tiếp từ vận chuyển thời gian thực trước khi nạp vào kho.
- Vấn đề tốc độ gán nhãn
    - Tốc độ gán nhãn là điểm hạn chế, đặc biệt đối với nhiệm vụ học liên tục với chu kỳ phản hồi ngắn.
    - Giải pháp: Sử dụng kỹ thuật giám sát yếu, giám sát bán giám thị hoặc tính toán nhãn trực tiếp từ vận chuyển thời gian thực để tăng tốc độ gán nhãn. Thách thức đối mặt với tính toán liên tục từ sự kiện.

2.1.4.2 Đánh giá

- Việc áp dụng việc học tập liên tục và cập nhật mô hình càng thường xuyên thì càng có nhiều cơ hội để mô hình thất bại.
- Điều này có nghĩa là việc thử nghiệm các mô hình của bạn trước khi triển khai chúng cho nhiều đối tượng hơn là rất quan trọng. Việc kiểm tra cần có thời gian

nên đây có thể là một yếu tố hạn chế khác về tần suất cập nhật mô hình nhanh nhất mà bạn có thể nhận được

2.1.4.3 Quy mô dữ liệu

- Tính toán đặc trưng thường đòi hỏi quy mô. Quy mô yêu cầu truy cập vào thống kê dữ liệu toàn cầu như giá trị nhỏ nhất, lớn nhất, trung bình và phương sai.
- Một kỹ thuật phổ biến để thực hiện việc này là tính toán hoặc ước tính các thống kê này tăng dần khi bạn quan sát dữ liệu mới (ngược lại với việc tải toàn bộ tập dữ liệu vào thời gian huấn luyện và tính toán từ đó).
    - Kĩ thuật "Optimal Quantile Approximation in Streams"
    - StandardScaler trong Scikit-learn có partial_fit cho phép sử dụng bộ tỷ lệ đặc trưng với thống kê đang chạy. Tuy nhiên, các phương pháp tích hợp sẵn này chậm và không hỗ trợ nhiều loại thống kê đang chạy.

2.1.4.4 Thuật toán

- Thách thức này xuất hiện khi bạn sử dụng một số loại thuật toán và muốn cập nhật chúng rất nhanh chóng. Những thuật toán cụ thể này được thiết kế để dựa vào việc truy cập toàn bộ bộ dữ liệu để được huấn luyện. Ví dụ: matrix-based, dimensionality reduction-based and tree-based models. Những loại mô hình này không thể được huấn luyện tăng dần với dữ liệu mới như các mô hình dựa trên trọng số như mạng nơ-ron có thể.
- Thách thức chỉ xuất hiện khi bạn cần cập nhật chúng rất nhanh chóng vì bạn không thể đợi thuật toán xử lý toàn bộ bộ dữ liệu.

2.1.5 Các Giai đoạn của Continual Learning

2.1.5.1 Manual, stateless retraining

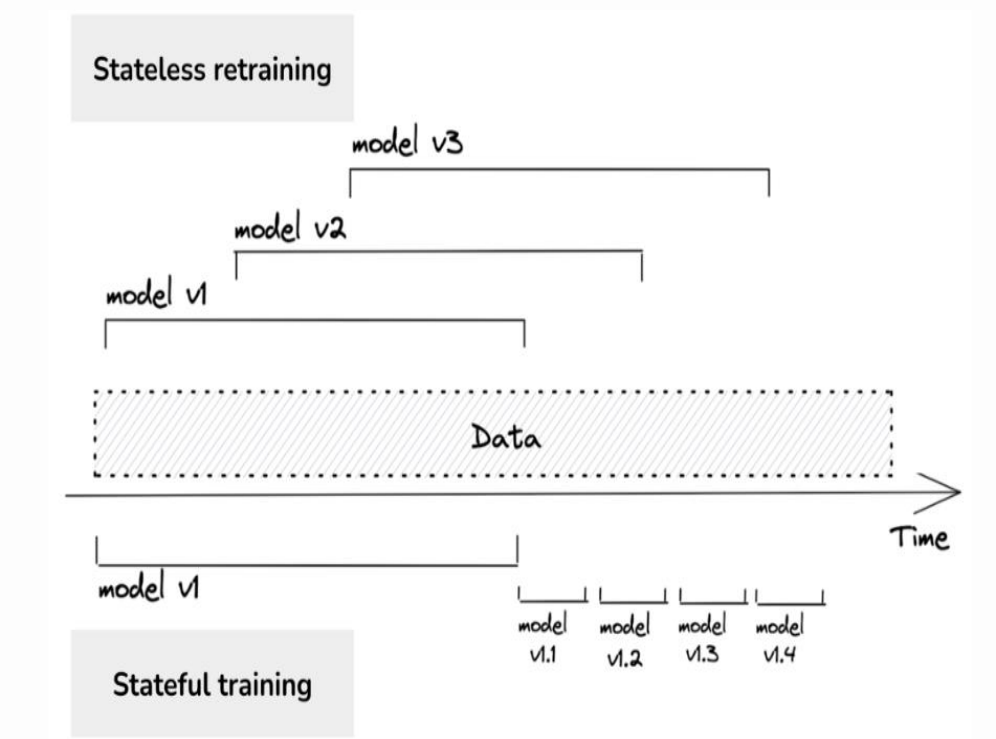
- Mô hình chỉ được huấn lại khi
    - Hiệu suất của mô hình đã giảm đến mức nó đang gây hại hơn là có ích
    - Đội ngũ của bạn có thời gian để cập nhật nó

2.1.5.2 Fixed schedule automated stateful training

- Thường xảy ra khi các mô hình chính của một lĩnh vực đã được phát triển và ưu tiên của bạn không phải là tạo ra mô hình mới mà là duy trì và cải tiến các mô hình hiện tại.
- Các hoạt động giai đoạn này:
    - Kéo dữ liệu.
    - Tải Xuống mẫu hoặc lấy mẫu dữ liệu nếu cần thiết.
    - Trích xuất features.
    - Xử lý và/hoặc chú thích nhãn để tạo dữ liệu huấn luyện.
    - Bắt đầu quá trình huấn luyện.
    - Đánh giá mô hình mới.
    - Triển khai nó.

2.1.5.3 Fixed schedule automated stateful training

- Để đạt được điều này, bạn cần điều chỉnh kịch bản của mình và một cách để theo dõi dữ liệu và hệ thống liên kết của mô hình. Điển hình là việc sử dụng phiên bản liên kết mô hình để theo dõi quá trình đào tạo và phiên bản của mô hình.
- Hãy nhớ rằng đào tạo trạng thái là khi bạn tiếp tục đào tạo mô hình của mình trên dữ liệu mới thay vì đào tạo lại mô hình của bạn từ đầu. Đào tạo lại trạng thái cho phép bạn cập nhật mô hình của mình với ít dữ liệu hơn



Hình 2. 2 Stateless retraining vs Stateful training

2.1.5.4 Continual learning

- Thay vì cập nhật mô hình của bạn dựa trên lịch trình cố định, hãy liên tục cập nhật mô hình của bạn bất cứ khi nào phân phối dữ liệu thay đổi và hiệu suất của mô hình giảm mạnh.
- Các yêu cầu cần có khi ở giai đoạn này:
    - Một cơ chế để kích hoạt cập nhật mô hình. Trình kích hoạt này có thể dựa trên thời gian (ví dụ: cứ sau 5 phút), dựa trên hiệu suất (ví dụ: bất cứ khi nào hiệu suất mô hình giảm mạnh) hoặc dựa trên độ trôi (ví dụ: bất cứ khi nào phân phối dữ liệu thay đổi).
    - Hầu hết các giải pháp giám sát ngày nay đều tập trung vào việc phân tích các tính năng – phân tích số liệu thống kê tóm tắt của một tính năng (ví dụ: trung bình, phương sai, tối thiểu, tối đa) và cảnh báo cho bạn khi có những thay đổi đáng kể trong các số liệu thống kê này. Tuy nhiên, một mô hình có thể có hàng trăm, thậm chí hàng nghìn tính năng. Hầu hết các thay đổi

thống kê tính năng đều lành tính. Vấn đề không phải là làm thế nào để phát hiện những thay đổi này mà là làm thế nào để biết thay đổi nào thực sự cần sự chú ý của bạn.

- Những cách tốt hơn để liên tục đánh giá mô hình của bạn.
- Một trình điều phối tự động khởi tạo các phiên bản nhằm cập nhật và đánh giá mô hình của bạn mà không làm gián đoạn dịch vụ dự đoán hiện có.

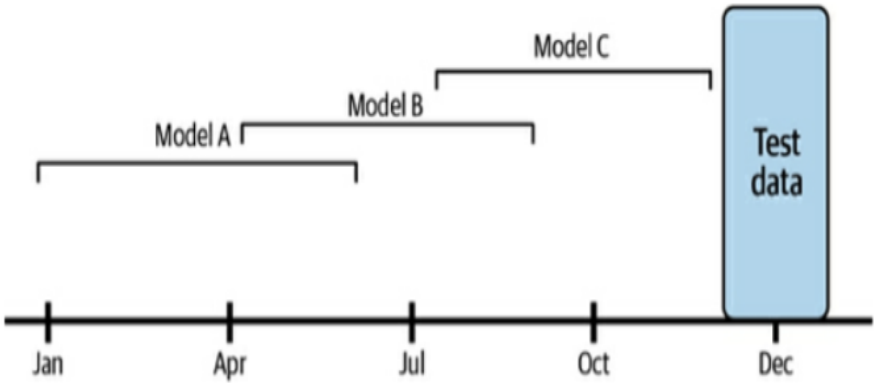
2.1.6 Tần suất cập nhật mô hình

– Cần xác định lợi ích nhận được khi cập nhật mô hình của mình với dữ liệu mới. Càng đạt được nhiều thì càng phải huấn luyện lại thường xuyên.

2.1.6.1 Đo lường giá trị của độ mới dữ liệu

– 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 (hình ảnh bên dưới).

– 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.



Hình 2. 3 Đo lường giá trị

2.1.6.2 Khi nào huấn luyện lại mô hình

– Trường hợp nên và không nên huấn luyện lại

- Nếu bạn tiếp tục giảm kích hoạt huấn luyện 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.
- Nếu chuyển sang một kiến trúc mô hình lớn hơn yêu cầu nhiều lần công suất tính toán mang lại cải thiện hiệu suất 1%, nhưng việc giảm thời gian kích hoạt huấn luyện lại xuống 3 giờ cũng mang lại cải thiện hiệu suất 1% với công suất tính toán chỉ 1 lần, hãy ưu tiên lặp lại dữ liệu thay vì lặp lại mô hình.
- Câu hỏi "khi nào thực hiện lặp lại mô hình và lặp lại dữ liệu". Cần sẽ phải chạy thử nghiệm nhiệm vụ cụ thể để tìm ra thời điểm thực hiện việc đó.

2.2 Testing models in Production

- Test production trong machine learning là quá trình kiểm thử và triển khai mô hình machine learning trong môi trường sản xuất.
- Để kiểm tra đầy đủ các mô hình của bạn trước khi phổ biến rộng rãi, bạn cần cả đánh giá offline trước khi triển khai và thử nghiệm trong sản xuất.
- 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 Pre-deployment offline evaluations

- Hai phương pháp phổ biến nhất:
  - Sử dụng một bộ dữ liệu thử nghiệm để so sánh với một mô hình cơ sở
  - Thực hiện các bài kiểm tra lại (running backtests)
- Bộ dữ liệu thử nghiệm thường là cố định để bạn có một để so sánh giữa nhiều mô hình. Điều này cũng có nghĩa là hiệu suất tốt trên một bộ dữ liệu thử nghiệm cố

định và cũ không đảm bảo hiệu suất tốt trong điều kiện phân phối dữ liệu hiện tại trong môi trường sản xuất.

- Kiểm tra lại (Backtesting) là ý tưởng sử dụng dữ liệu được gán nhãn mới nhất mà mô hình chưa nhìn thấy trong quá trình huấn luyện để kiểm tra hiệu suất.

2.2.2 Testing in Production Strategies

2.2.2.1 Shadow Deployment

- Triển khai mô hình mới song song với mô hình hiện tại. Chuyển mọi yêu cầu đến cả hai mô hình nhưng chỉ phục vụ kết quả dự đoán của mô hình hiện tại. Lưu lại các dự đoán từ cả hai mô hình để so sánh sau này.
- Ưu điểm:
    - Đây là cách triển khai an toàn nhất cho mô hình của bạn. Ngay cả khi mô hình mới của bạn có lỗi, dự đoán sẽ không được phục vụ.
    - Khá đơn giản về mặt khái niệm.
    - Thử nghiệm của bạn sẽ thu thập đủ dữ liệu để đạt đến ý nghĩa thống kê nhanh chóng hơn so với tất cả các chiến lược khác vì tất cả các mô hình đều nhận toàn bộ lưu lượng.
- Nhược điểm:
    - Kỹ thuật này không thể sử dụng khi đo lường hiệu suất của mô hình phụ thuộc vào việc quan sát cách người dùng tương tác với các dự đoán.
    - Kỹ thuật này tốn kém vì nó làm tăng gấp đôi số lượng dự đoán và do đó làm tăng gấp đôi công suất tính toán yêu cầu.

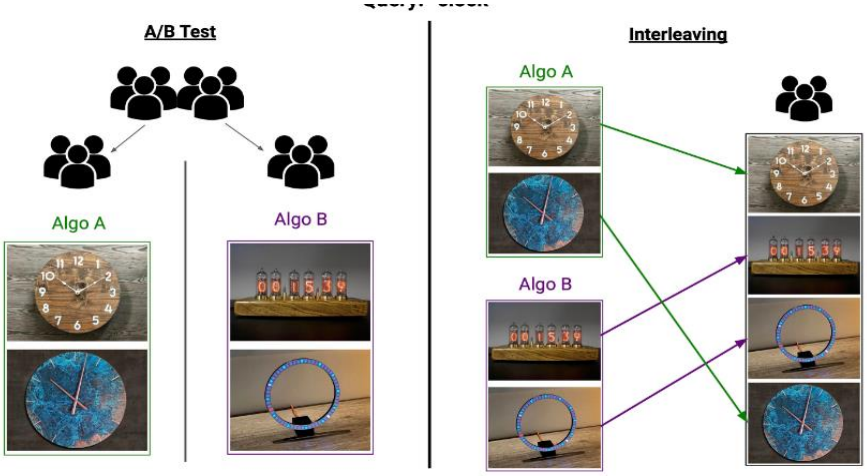
2.2.2.2 A/B Testing

- Thử nghiệm A/B là một phương pháp thiết yếu cho machine learning vì nó có thể giúp xác thực và cải thiện các mô hình trong môi trường thực tế, so sánh và tối ưu hóa các mô hình, thuật toán, thông số và tính năng khác nhau, đo lường tác động của các mô hình machine learning lên sản phẩm hoặc thiết kế, học hỏi và lặp lại từ dữ liệu và phản hồi

- A/B testing for machine learning deployment
    - Sử dụng A/B Testing để kiểm tra và hiệu chỉnh việc triển khai của mô hình học máy mới.
    - Đánh giá liệu mô hình mới có cải thiện so với mô hình hiện tại không.
- Automating A/B testing with machine learning
    - Sử dụng học máy để tự động hóa quá trình A/B Testing, giảm chi phí nguồn lực của các thử nghiệm liên tục hoặc nhiều lần.
    - Các thuật toán học máy giúp tối ưu hóa quy trình A/B Testing, có nhiều sản phẩm sẵn có.
- Discovering information about datasets with A/B testing
    - Sử dụng A/B Testing để xác định mối quan hệ trong bộ dữ liệu, hiểu cách thay đổi một biến có thể ảnh hưởng đến một kết quả.
    - Kết quả của A/B Testing cung cấp thông tin rõ ràng từ môi trường thực tế động, hữu ích khi xem xét mô hình học máy.

2.2.2.3 Canary Release

- Chiến lược Canary deployment tập trung vào việc phát triển và triển khai phiên bản mới của ứng dụng ở quy mô nhỏ, an toàn và kiểm thử trước khi đưa ra cho người dùng cuối.
- Trong chiến lược Canary deployment, phiên bản mới của ứng dụng được triển khai cùng lúc với phiên bản hiện tại. Tuy nhiên, chỉ một phần nhỏ người dùng được chuyển hướng sử dụng phiên bản mới, trong khi phần lớn vẫn sử dụng phiên bản hiện tại.
- Qua quá trình theo dõi, người quản lý có thể thu thập thông tin về hiệu suất và ổn định của phiên bản mới. Nếu phiên bản Canary không gặp vấn đề và đạt được các chỉ số hiệu suất mong đợi, thì có thể tiến hành chuyển hướng toàn bộ người dùng sang phiên bản mới. Ngược lại, nếu có vấn đề, ngừng triển khai hoặc chuyển hướng trở lại phiên bản hiện tại để tránh tác động đến người dùng.

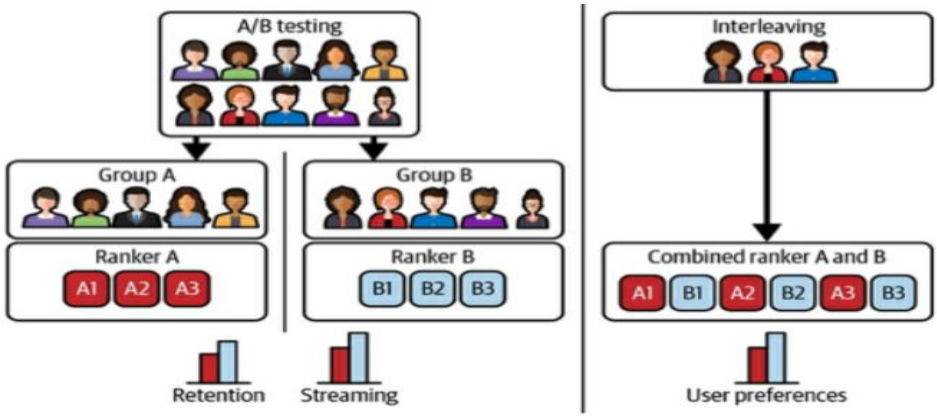


Hình 2. 4 So sánh A/B vs Interleaving

– Ưu điểm
  - Dễ hiểu và đơn giản.
  - Đơn giản nhất trong tất cả các chiến lược để triển khai nếu bạn đã có cơ sở hạ tầng đặc điểm chức năng trong công ty.
  - Vì các dự đoán của mô hình thử thách sẽ được phục vụ, bạn có thể sử dụng điều này với các mô hình yêu cầu tương tác của người dùng để thu thập hiệu suất.
  - So với triển khai đổ bóng (shadow deployments), nó rẻ hơn để chạy với một dự đoán mỗi yêu cầu.
  - Nếu kết hợp với kiểm thử A/B, nó cho phép bạn động động thay đổi lượng lưu lượng mà mỗi mô hình đang nhận.
– Nhược điểm:
  - Khả năng không chặt chẽ trong việc xác định sự khác biệt về hiệu suất.
  - Nếu các phiên bản không được giám sát cẩn thận, tai nạn có thể xảy ra. Điều này có thể được coi là lựa chọn ít an toàn nhất, nhưng lại rất dễ quay trở lại (rollback).

2.2.2.4 Interleaving Experiments

– Trong thử nghiệm A/B, một người dùng sẽ nhận được dự đoán từ mô hình A hoặc mô hình B. Khi xen kẽ, một người dùng sẽ nhận được dự đoán xen kẽ từ cả mô hình A và mô hình B. Sau đó, chúng tôi theo dõi hoạt động của từng mô hình bằng cách đo lường tùy chọn của người dùng với dự đoán của từng mô hình (ví dụ: người dùng nhấp nhiều hơn vào đề xuất từ mô hình B)



Hình 2. 5 So sánh A/B vs Interleaving

– Ưu điểm

- Việc xen kẽ độc lập có thể xác định mô hình tốt nhất với kích thước mẫu đáng kể nhỏ hơn so với kiểm thử A/B truyền thống.
- Chiến lược này cho phép bạn theo dõi cách người dùng tương tác với dự đoán của bạn

– Nhược điểm

- Việc triển khai phức tạp hơn so với kiểm thử A/B.
- Bạn cần lo lắng về trường hợp đặc biệt về việc xử lý nếu một trong những mô hình xen kẽ mất quá nhiều thời gian để phản hồi hoặc gặp sự cố.
- Điều này làm tăng gấp đôi năng lực tính toán cần thiết vì mỗi yêu cầu đều nhận các dự đoán từ nhiều mô hình.
- Nó không dễ mở rộng cho một số lượng lớn mô hình thách thức. 2-3 mô hình xen kẽ dường như là điểm đáng chú ý.

2.2.2.5 Bandits

- Bandits là một thuật toán theo dõi hiệu suất hiện tại của từng biến thể mô hình và đưa ra quyết định linh hoạt đối với mọi yêu cầu về việc nên sử dụng mô hình có hiệu suất cao nhất cho đến nay (tức là khai thác kiến thức hiện tại) hay thử bất kỳ mô hình nào trong số đó. các mô hình khác để có thêm thông tin về chúng (tức là khám phá xem một trong các mô hình khác thực sự tốt hơn).
- Có rất nhiều thuật toán bandits. Đơn giản nhất được gọi là epsilon greedy. Hai công cụ mạnh mẽ và phổ biến nhất là Thompson Sampling và Upper Confidence Bound (UCB)
- Ưu điểm
    - Bandits yêu cầu ít dữ liệu hơn so với A/B testing để xác định mô hình nào tốt hơn
    - Bandits có hiệu suất dữ liệu cao hơn đồng thời giảm thiểu chi phí cơ hội của bạn. Trong nhiều trường hợp, bandit được coi là tối ưu.
    - So với A/B testing, bandits an toàn hơn vì nếu một mô hình thực sự tồi, thuật toán sẽ chọn nó ít hơn. Hơn nữa, sự hội tụ sẽ nhanh chóng giúp bạn loại bỏ thách thức tồi.
- Nhược điểm:
    - So với tất cả các chiến lược khác, bandits khó triển khai hơn rất nhiều do cần phải liên tục lan truyền phản hồi vào thuật toán.
    - Bandits chỉ có thể được sử dụng trong một số trường hợp sử dụng cụ thể.
    - Chúng không an toàn như Shadow Deployments vì thách thức nhận lưu lượng trực tiếp.

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Optimizer- Hiểu sâu về các thuật toán tối ưu
2. Thuật toán Adam
3. Tối ưu thuật toán

Tiếng Anh

1. **Optimization in Deep Learning: AdaGrad, RMSProp, ADAM**
2. **An overview of gradient descent optimization algorithms**
3. **Optimization: Stochastic Gradient Descent**
4. **Continual learning and test in production**
5. **A/B Testing for Machine Learning**