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

NGUYỄN THANH TÚ - 52100349

BÁO CÁO CÁ NHÂN CUỐI KÌ NHẬP MÔN HỌC MÁY

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

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

NGUYỄN THANH TÚ - 52100349

BÁO CÁO CÁ NHÂN CUỐI KÌ NHẬP MÔN HỌC MÁY

Người hướng dẫn
PGS.TS. Lê Anh Cường

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn đến toàn thể giảng viên trường Đại học Tôn Đức Thắng nói chung và giảng viên Lê Anh Cường Nói riêng vì đã tạo môi trường học tập, phát triển toàn diện cho chúng em. Ngoài ra, trong suốt quá trình học tập chúng em đã được dạy và tiếp thu nhiều kiến thức bổ ích, quan trọng đối với tương lai của chúng em.

Bằng kiến thức mà chúng em tiếp thu được trong suốt quá trình học tập tại trường để hoàn thiện bài báo cáo này. Tuy còn nhiều thiếu sót cần phải học hỏi và cải thiện trong bài báo cáo nên chúng em mong rằng sẽ được thầy nhận xét, đánh giá và cho ý kiến về bài báo cáo.

Chúng em rất mong có được sự góp ý đến từ thầy cô tại trường để biết được những thiếu sót của bản thân đồng thời cải thiện kỹ năng của chúng em. Cuối cùng, chúng em xin gửi lời cảm ơn chân thành đến quý thầy cô trường Đại học Tôn Đức Thắng và thầy Lê Anh Cường.

TP. Hồ Chí Minh, ngày 1 tháng 12 năm 2023

Tác giả

(Ký tên và ghi rõ họ tên)

Nguyễn Thanh Tú

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH

TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Chúng em xin cam đoan đây là đồ án của riêng nhóm chúng em và được sự hướng dẫn khoa học của PGS.TS Lê Anh Cường. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chúng em 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 tôi xin hoàn toàn chịu trách nhiệm về nội dung Báo cáo của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 1 tháng 12 năm 2023

Tác giả

(Ký tên và ghi rõ họ tên)

Nguyễn Thanh Tú

TÓM TẮT

Trong bài báo cáo này, chúng tôi đã tìm hiểu và so sánh các phương pháp optimizer trong huấn luyện mô hình học máy, cũng như tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.Các phương pháp Optimizer.

Optimizer là một thuật toán được sử dụng để tìm ra các tham số tối ưu của một mô hình học máy. Có nhiều phương pháp optimizer khác nhau, mỗi phương pháp có những ưu nhược điểm riêng.

Continual Learning là một lĩnh vực nghiên cứu học máy tập trung vào việc đào tạo các mô hình học máy có thể học hỏi và thích nghi với dữ liệu mới mà không bị quên đi những gì đã học trước đó.

Continual Learning là một thách thức khó khăn vì các mô hình học máy thường dễ bị quên đi những gì đã học trước đó khi được đào tạo trên dữ liệu mới. Có nhiều kỹ thuật khác nhau có thể được sử dụng để giải quyết thách thức này.

Test Production là một quá trình liên tục kiểm tra và triển khai các mô hình học máy mới trong môi trường sản xuất.Test Production giúp đảm bảo rằng các mô hình học máy mới vẫn hoạt động tốt trong môi trường sản xuất và đáp ứng các yêu cầu kinh doanh.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC CÁC CHỮ VIẾT TẮT

NAG	Nesterov accelerated gradient
SGD	Stochastic Gradient Descent
GD	Gradient Descent
GEM	Gradient Episodic Memory
RMSprop	Root Mean Square Propagation

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

Trong quá trình huấn luyện mô hình học máy, một bước quan trọng là tối ưu hóa các tham số của mô hình để giảm thiểu hàm mất mát. Các phương pháp Optimizer chịu trách nhiệm điều chỉnh các tham số này dựa trên thông tin được tính toán từ đạo hàm của hàm mất mát.

Trong học máy, các thuật toán tối ưu hóa được sử dụng để tìm các tham số của mô hình phù hợp nhất với tập dữ liệu huấn luyện. Có nhiều phương pháp tối ưu hóa khác nhau, mỗi phương pháp có cách thức hoạt động và ưu điểm riêng.

Gradient Descent: là một phương pháp tối ưu hóa đơn giản nhưng hiệu quả. Phương pháp này liên tục cập nhật các tham số của mô hình theo hướng giảm hàm mất mát.

Momentum: là một kỹ thuật cải tiến Gradient Descent. Phương pháp này sử dụng một biến động lượng để giúp mô hình vượt qua các điểm dừng cục bộ.

Nesterov accelerated gradient (NAG): là một biến thể của Momentum. Phương pháp này có thể giúp mô hình hội tụ nhanh hơn.

Stochastic Gradient Descent(SGD): là một biến thể của Gradient Descent. Phương pháp này chỉ sử dụng một mẫu ngẫu nhiên từ tập dữ liệu huấn luyện để cập nhật các tham số của mô hình.

Stochastic Gradient Descent with Gradient Clipping: là một biến thể của Stochastic Gradient Descent. Phương pháp này hạn chế độ lớn của gradient để tránh quá trình đào tạo bị trượt khỏi vùng tối ưu.

Adagrad: là một phương pháp tối ưu hóa bậc một khác. Phương pháp này điều chỉnh tốc độ học cho từng tham số của mô hình.

RMSprop: là một phương pháp tối ưu hóa bậc một dựa trên ý tưởng cân nhắc trọng số của các gradient trong quá trình cập nhật tham số.

Adam: là một phương pháp tối ưu hóa bậc một hiệu quả. Phương pháp này kết hợp các ưu điểm của Momentum và RMSprop.

AdamW: là một biến thể của Adam. Ý tưởng của AdamW khá đơn giản: khi thực hiện thuật toán Adam với L2 regularization

Để tìm hiểu về những thuật toán kể trên, đầu tiên ta cần tìm hiểu về thuật toán **Gradient Descent** cơ bản:

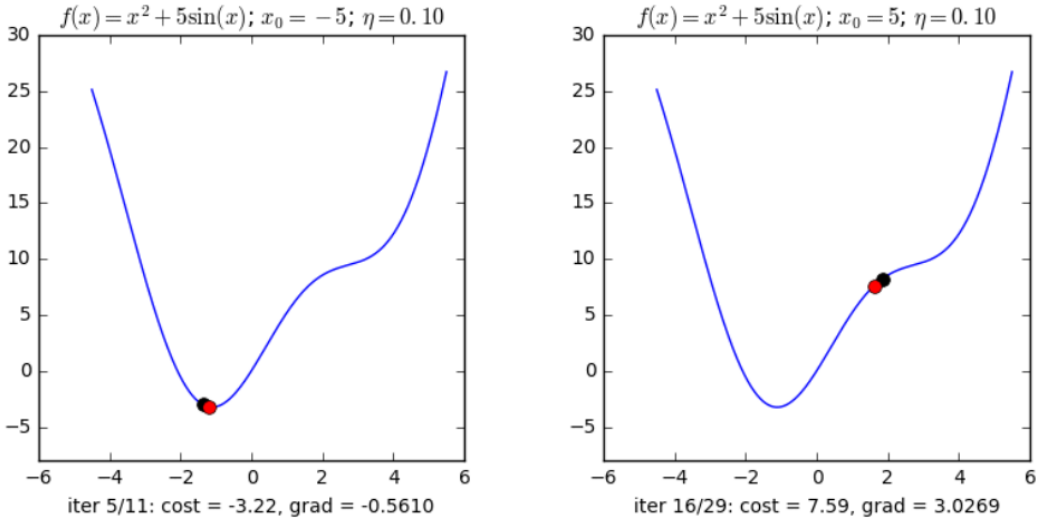
1.1 Gradient Descent (GD)

Trong các bài toán tối ưu, chúng ta thường tìm giá trị nhỏ nhất của 1 hàm số nào đó, mà hàm số đạt giá trị nhỏ nhất khi đạo hàm bằng 0. Nhưng không phải lúc nào đạo hàm hàm số cũng được, đối với các hàm số nhiều biến thì đạo hàm rất phức tạp, thậm chí là bất khả thi. Nên thay vào đó người ta tìm điểm gần với điểm cực tiểu nhất và xem đó là nghiệm bài toán. Gradient Descent có hướng tiếp cận ở đây là chọn 1 nghiệm ngẫu nhiên cứ sau mỗi vòng lặp (hay epoch) thì cho nó tiến dần đến điểm cần tìm.

- Gradient Descent cho hàm 1 biến:
  Công thức cho Gradient Descent (GD):

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

  Trong đó, η là một số dương được gọi là learning rate(tốc độ học). Dấu trừ thể hiện chúng ta phải đi ngược với đạo hàm(Đây cũng là lý do phương pháp này được gọi là Gradient Descent - descent nghĩa là đi ngược) .
  Ví dụ, như đối với hàm f(x)=2x+5*sin(x) và f’(x) = 2x+5*cos(x)
  với x_old=-4 thì f’(-4) <0 => x_new > x_old nên nghiệm sẽ di chuyển về bên phải tiến gần cực tiểu cực tiểu. Ngược lại với x_old =4 thì f’(4)>0 => x_new< x_old nên nghiệm sẽ di chuyển về bên trái tiến gần tới điểm cực tiểu.



Hình 1 :Ví dụ cho Gradient Descent một biến

- Gradient Descent cho hàm nhiều biến

Giả sử ta cần tìm global minimum cho hàm f(θ) trong đó θ (theta) là một vector, thường được dùng để ký hiệu tập hợp các tham số của một mô hình cần tối ưu (trong Linear Regression thì các tham số chính là hệ số w). Đạo hàm của hàm số đó tại một điểm θ bất kỳ được ký hiệu là $\nabla_\theta f(\theta)$(hình tam giác ngược đọc là nabla). Tương tự như hàm 1 biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán θ_0, sau đó, ở vòng lặp thứ t, quy tắc cập nhật là:

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

Hoặc đơn giản là :$\theta = \theta - \eta\nabla_\theta f(\theta)$

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.
- 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.

1.2 Gradient Descent với momentum

Để khắc phục các hạn chế trên của thuật toán Gradient Descent người ta dùng gradient descent with momentum. Gradient Descent với Momentum cải tiến GD bằng cách tích lũy một vector momentum để giúp việc cập nhật trọng số đạt được đà tốt hơn. Momentum có thể được hiểu như khối lượng của bước di chuyển và giúp giảm độ dao động trong quá trình tối ưu hóa. Trong GD, chúng ta cần tính lượng thay đổi ở thời điểm t để cập nhật vị trí mới cho nghiệm. Nếu chúng ta coi đại lượng này như vận tốc v_t trong vật lý, nghiệm mới 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 v_t sao cho nó vừa mang thông tin của đạo hàm, vừa mang thông tin của đà, tức là vận tốc trước đó v_{t-1}(chúng ta coi vận tốc ban đầu v0 =0). Một cách đơn giản hơn nhất, ta có thể cộng (có trọng số) hai đại lượng này lại:

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

Trong đó, γ thường được chọn là 1 giá trị khoảng 0.9, v_t là vận tốc hãy đạo hàm tại thời điểm trước đó, $\nabla_\theta J(\theta)$ chính là độ dốc của điểm trước đó. Sau đó vị trí nghiệm mới được xác định như sau:

$$\theta = \theta - v_t$$

Thuật toán đơn giản này tỏ ra rất hiệu quả trong các bài toán thực tế (trong không gian nhiều chiều, cách tính toán cũng hoàn toàn tương tự).

Ưu điểm :

Tăng tốc độ hội tụ: Momentum giúp tăng tốc độ hội tụ của quá trình huấn luyện bằng cách tích lũy đạo hàm của các bước trước đó. Điều này giúp giảm tình trạng dao động và giúp mô hình vượt qua các điểm cực tiểu cục bộ.

Vượt qua vùng phẳng: Khi mô hình đạt đến vùng phẳng trong không gian tối ưu, Momentum giúp mô hình vượt qua các điểm tối thiểu địa phương bằng cách giảm độ lớn của gradient và tăng tốc độ di chuyển.

Ổn định hơn với dữ liệu nhiễu: Momentum cũng có khả năng giảm ảnh hưởng của dữ liệu nhiễu trong quá trình huấn luyện. Bằng cách tích lũy đạo hàm từ các bước trước đó, nó có khả năng giảm thiểu tác động của các điểm dữ liệu nhiễu đơn lẻ.

Nhược điểm :

Độ lớn của gradient: Một nhược điểm của Momentum là nếu độ lớn của gradient quá lớn, nó có thể khiến quá trình tối ưu hóa bị dao động

hoặc không hội tụ. Điều này đặc biệt đáng chú ý khi áp dụng Momentum với một learning rate lớn.

Khả năng trượt quá điểm tối ưu: Trong một số trường hợp, Momentum có thể vượt quá điểm tối ưu và tiếp tục di chuyển xa điểm tối ưu hóa mong muốn. Điều này có thể xảy ra khi learning rate lớn hoặc khi mô hình gần đạt tới điểm tối ưu.

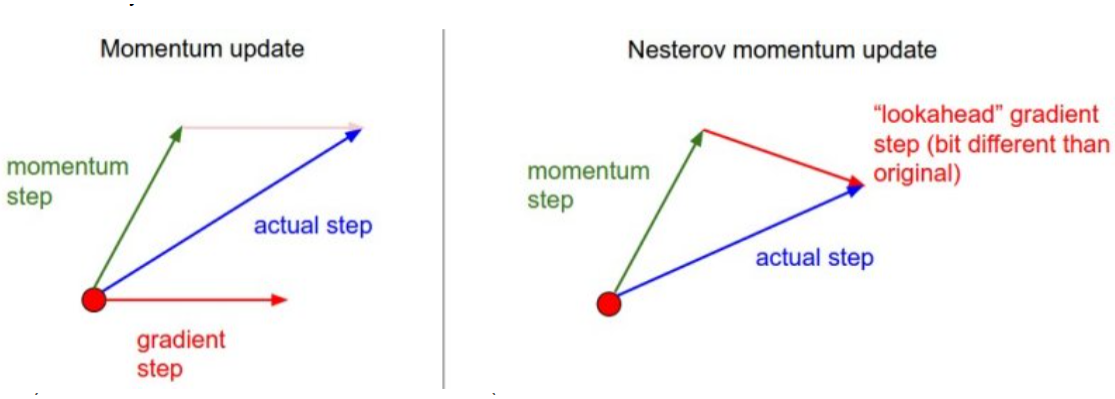
Điều chỉnh learning rate: Momentum không điều chỉnh tỉ lệ học tự động, do đó cần phải điều chỉnh learning rate một cách thủ công. Nếu không điều chỉnh đúng, nó có thể dẫn đến tình trạng quá tối ưu hoặc không hội tụ.

Phụ thuộc vào định dạng dữ liệu: Momentum có thể hoạt động không tốt trên một số loại dữ liệu hoặc kiến trúc mô hình cụ thể. Hiệu suất của Momentum có thể thay đổi tùy thuộc vào đặc điểm của bài toán và dữ liệu.

1.3 Nesterov accelerated gradient (NAG)

Có một của hạn chế momentum chúng ta có thể thấy trong ví dụ trên: Khi tới gần đích, momentum vẫn mất khá nhiều thời gian trước khi dừng lại. Lý do lại cũng chính là vì có đà. Có một phương pháp khác tiếp tục giúp khắc phục điều này, phương pháp đó mang tên Nesterov accelerated gradient (NAG), giúp cho thuật toán hội tụ nhanh hơn.Nesterov Momentum là một biến thể khác của Momentum, được mở rộng từ Gradient Descent với Momentum. Nó cải thiện Momentum bằng cách tính toán độ dốc của hàm mất mát tại một vị trí trước khi cập nhật trọng số. Điều này giúp Momentum "nhìn trước" và điều chỉnh hướng di chuyển theo đúng hướng.

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 nghiệm là $\theta - \gamma v_{t-1}$(chúng ta không đính kèm phần gradient ở đây vì sẽ sử dụng nó trong bước cuối cùng). Vậy, thay vì sử dụng gradient của điểm hiện tại, NAG đi trước một bước, sử dụng gradient của điểm tiếp theo. Theo dõi hình dưới đây:

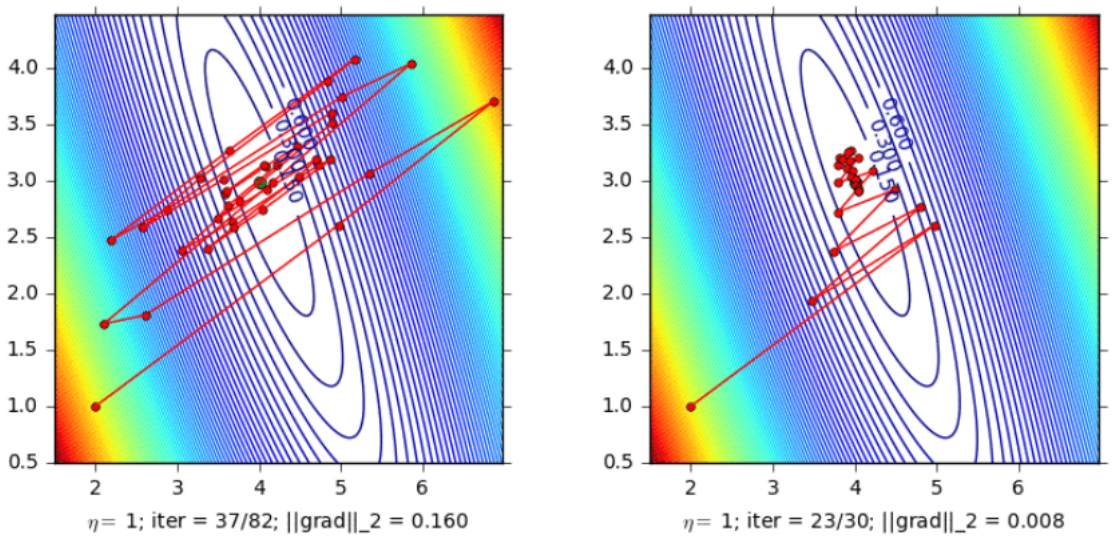


Hình 2 :Nesterov accelerated gradient

- 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.

Công thức cập nhật của NAG được cho như sau:

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



Hình 3 :Ví dụ so sánh Momentum và NAG cho bài toán Linear Regression

Hình bên trái là đường đi của nghiệm với phương pháp Momentum. nghiệm đi khá là *zigzag* và mất nhiều vòng lặp hơn. Hình bên phải là đường đi của nghiệm với phương pháp NAG, nghiệm hội tụ nhanh hơn, và đường đi ít *zigzag* hơn.

1.4 Stochastic Gradient Descent (SGD):

Stochastic Gradient Descent (SGD) là một phương pháp tối ưu hóa phổ biến trong lĩnh vực học máy. Nó là một biến thể của Gradient Descent (GD) và được sử dụng rộng rãi để huấn luyện các mô hình học máy trên các tập dữ liệu lớn.

SGD khác với GD ở cách tính toán độ dốc và cập nhật trọng số. Trong GD, toàn bộ tập dữ liệu được sử dụng để tính toán độ dốc và cập nhật trọng số mỗi lần cập nhật. Trong khi đó, SGD chỉ sử dụng một mẫu dữ liệu ngẫu nhiên (một điểm dữ liệu) để tính toán độ dốc và cập nhật trọng số. Điều này giúp SGD tiết kiệm thời gian tính toán, đặc biệt là trên các tập dữ liệu lớn.

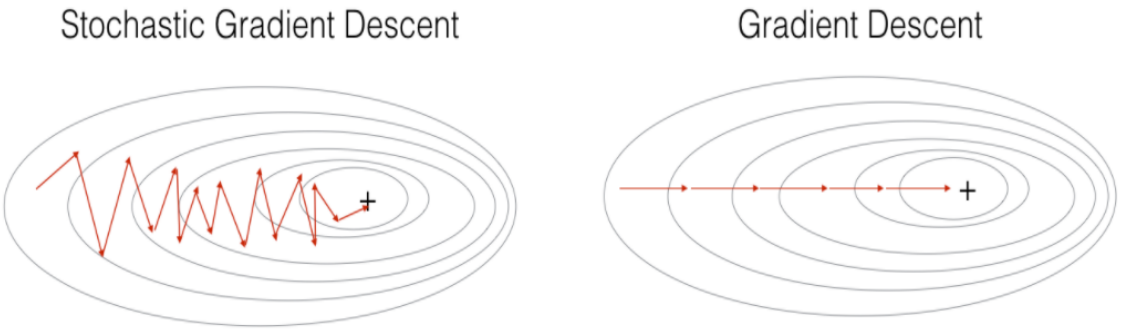
Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với GD thông thường thì mỗi epoch ứng với 1 lần cập nhật θ, với SGD thì mỗi epoch ứng với N lần cập nhật θ với N là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu một lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). Vì vậy SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn.

Một điểm cần lưu ý đó là: sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên. Việc này cũng ảnh hưởng tới hiệu năng của SGD.

Một cách toán học, quy tắc cập nhật của SGD là:

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

Trong đó, $J(\theta; x_i; y_i)$ là hàm mất mát với chỉ 1 cặp điểm dữ liệu (input, label) là $(x_i; y_i)$. Ta có thể hoàn toàn áp dụng các thuật toán tăng tốc GD như Momentum, AdaGrad,... vào SGD.



Hình 4: So sánh đường đi của Stochastic Gradient với Gradient Descent

Nhìn vào 2 hình trên, ta thấy SGD có đường đi khá là zig zắc , không mượt như GD. Dễ hiểu điều đó vì 1 điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu. Ở đây, GD có hạn chế đối với cơ sở dữ liệu lớn (vài triệu dữ liệu) thì việc tính toán đạo hàm trên toàn bộ dữ liệu qua mỗi vòng lặp trở nên cồng kềnh. Bên cạnh đó GD không phù hợp với online learning. **Online learning** là khi dữ liệu cập nhật liên tục (ví dụ như thêm người dùng đăng kí) thì mỗi lần thêm dữ liệu ta phải tính lại đạo hàm trên toàn bộ dữ liệu dẫn đến thời gian tính toán lâu, thuật toán không online nữa. Vì thế SGD ra đời để giải quyết vấn đề đó, vì mỗi lần thêm dữ liệu mới vào chỉ cần cập nhật trên 1 điểm dữ liệu đó thôi, phù hợp với online learning.
Một ví dụ minh hoạ : có 10.000 điểm dữ liệu thì chỉ sau 3 epoch ta đã có được nghiệm tốt, còn với GD ta phải dùng tới 90 epoch để đạt được kết quả đó.

Ưu điểm:

Hiệu quả tính toán: SGD chỉ cần tính toán độ dốc dựa trên một mẫu dữ liệu, giúp giảm thời gian tính toán so với GD khi làm việc với các tập dữ liệu lớn.
Khả năng làm việc với dữ liệu không đồng nhất: SGD có thể làm việc tốt với các tập dữ liệu không đồng nhất, trong đó các mẫu dữ liệu có độ quan trọng khác nhau.
Có thể điều chỉnh tỷ lệ học: SGD cho phép điều chỉnh tỷ lệ học (learning rate) tùy ý, giúp kiểm soát tốc độ hội tụ và tránh quá mức điều chỉnh trọng số.

Tuy nhiên, SGD cũng có một số nhược điểm:

Độ dao động cao: Do tính ngẫu nhiên của việc chọn mẫu dữ liệu, SGD có xu hướng dao động xung quanh giá trị tối ưu và không hội tụ một cách chính xác như GD.
Cần điều chỉnh tỷ lệ học thích hợp: Tỷ lệ học quá cao có thể khiến SGD bị phóng đại độ dốc và không hội tụ, trong khi tỷ lệ học quá thấp có thể làm chậm quá trình tối ưu hóa.

1.5 Adagrad

Trong một số trường hợp, việc chọn learning rate phù hợp có thể trở thành một vấn đề nhức nhối. Vấn đề này sẽ được giải quyết với AdaGrad, vì thuật toán này sẽ coi learning rate là một tham số biến thiên sau mỗi lần chạy thuật toán.

AdaGrad thường được dùng trong các vấn đề về sparse feature (đặc trưng thưa), ví dụ như các bài toán về xử lí ngôn ngữ tự nhiên, quảng

cáo điện toán, và lọc cộng tác. Thường những những đặc trưng này ít khi được cập nhật để có thể được tối ưu những lại mang những ý nghĩa vô cùng to lớn, còn những đặc trưng có tần số xuất hiện cao thì được cập nhật liên tục. Vì thế, việc có một learning rate cố định hoặc là quá cao đối với những feature được cập nhật nhiều lần hoặc là quá thấp đối với những feature ít được cập nhật.

Công thức của AdaGrad như sau:

$$g_t = \delta_w l(y_t, f(x_t, w)),$$

$$s_t = s_{t-1} + g_t^2,$$

$$W_t = W_{t-1} - \frac{\eta}{\sqrt{S_t + \epsilon}}.g_t$$

Trong đó,

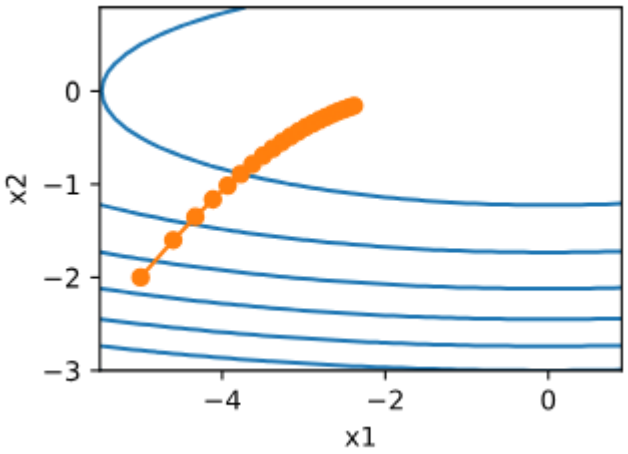
η: Tỷ lệ học ban đầu.

ε: Một giá trị nhỏ (thường là 10^{-8}) được thêm vào trong mẫu số để tránh chia cho 0.

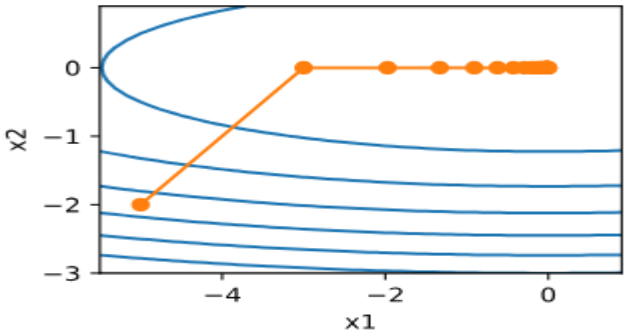
S_t là tổng các bình phương của gradient trước đó (t-1).

gt là gradient hiện tại của hàm mất mát l theo tham số θ.

Và, S_0thường được chọn bằng 0.



Hình 5:AdaGrad với lr=0.4



Hình 6: AdaGrad với lr=2

Ta có thể hiểu sự thay đổi learning rate này như một hệ số "ma sát", "ma sát" này có giá trị lớn ở dốc và có giá trị nhỏ ở vùng bằng phẳng.

Ưu điểm:

- Hoạt động tốt với bài toán sparse feature, cụ thể là tối ưu các bài toán lồi, không dao động nhiều khi đạt vào cực tiểu cần thiết.
- Learning rate được tự động điều chỉnh cho phù hợp
- Chi phí tính toán không tăng nếu so với SGD

Nhược điểm:

- Trong deep learning, các bài toán thường không hoàn toàn lồi và có nhiều local minimum gây nhiễu.
- s_n trong công thức cập nhật tăng một cách dường như tuyến tính khiến cho learning rate bị thu nhỏ nhanh.

1.6 RMSprop

AdaGrad với tốc độ học thay đổi để thích nghi với dữ liệu là một phương pháp tuyệt vời trong các bài toán tối ưu lồi, thế nhưng nó vẫn có những hạn chế như đã nêu trên. Vì thế RMSProp được đề xuất để khắc phục những vấn đề của AdaGrad bằng cách kế thừa ý tưởng từ phương pháp momentum trong gradient descent làm rò rỉ các giá trị của trong công thức cập nhật.Nói cách khác, Thuật toán RMSProp sử dụng một phiên bản điều chỉnh của AdaGrad để cập nhật tỷ lệ học động (learning rate) trong quá trình tối ưu hóa. Mục tiêu của RMSProp là giảm bớt ảnh hưởng của các bước cập nhật quá lớn và giữ cho tỷ lệ học động ổn định.

Dưới đây là công thức của thuật toán RMSProp:

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

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

Trong đó:

- $E[g^2]_t$ là tích lũy phương sai của các bình phương gradient trước đó (t-1).
- $\frac{\eta}{\sqrt{E[g^2]t+\epsilon}} g_t$ sự thay đổi tham số trong mô hình
- P là tham số suy giảm
- gt là gradient hiện tại của hàm mất mát J theo tham số θ. Với gt =∇J(θ).

Công thức trên cho thấy rằng RMSProp tính toán trung bình điều chỉnh của các bình phương gradient trước đó. Công thức này sử dụng hệ số giảm tỷ lệ học động ρ để điều chỉnh sự quan trọng của các bước cập nhật trước đó. Khi gradient thay đổi nhiều, trung bình điều chỉnh sẽ tăng lên. Ngược lại, khi gradient thay đổi ít, trung bình điều chỉnh sẽ giảm đi. Điều này giúp kiểm soát tỷ lệ học động và giảm thiểu sự dao động của quá trình tối ưu hóa.

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.

1.7 Adam

Adam (Adaptive Moment Estimation) là một thuật toán tối ưu hóa gradient phổ biến trong huấn luyện mạng nơ-ron sâu (deep learning). Thuật toán Adam kết hợp các ưu điểm của thuật toán RMSProp và thuật toán Momentum.
Như đã nói ở trên Adam là sự kết hợp của Momentum và RMSprop . Nếu giải thích theo hiện tượng vật lí thì Momentum giống như 1 quả cầu lao xuống dốc, còn Adam như 1 quả cầu rất nặng có ma sát, vì vậy nó dễ dàng vượt qua local minimum tới global minimum và khi tới global minimum nó không mất nhiều thời gian dao động qua lại quanh đích vì nó có ma sát nên dễ dừng lại hơn.
Như đã nói ở trên, Adam 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 mt . Các giá trị m_t và v_t được tính bởi công thức:
Đầu tiên, chúng ta cần khởi tạo các giá trị ban đầu:

- θ: Tham số mô hình cần cập nhật.
- α: Tỷ lệ học động (learning rate).
- β_1: Hệ số giảm gradient thứ nhất (first moment decay rate) (thường được đặt là 0.9).
- β_2: Hệ số giảm chuẩn thứ hai (second moment decay rate) (thường được đặt là 0.999).
- ε: Một giá trị nhỏ (thường là 10^{-8}) được thêm vào trong mẫu số để tránh chia cho 0.

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 β1 và β2 là các trọng số không âm, thường được chọn là β1 = 0.9 và β2 = 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 β1 và β2 xấp xỉ bằng 1. Do vậy, để khắc phục, các giá trị này được ước lượng bằng cách:

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

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

Sau đó cập nhật các trọng số theo công thức:

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

Trong đó:

- g là gradient hiện tại của hàm mất mát J theo tham số θ.
- m_t là bước chuẩn hóa thứ nhất, tính tổng gia tốc gradient trước đó với hệ số giảm gradient β_1.
- v_t là bước chuẩn hóa thứ hai, tính tổng bình phương gradient trước đó với hệ số giảm chuẩn β_2.
- t là chỉ số bước huấn luyện hiện tại.

Công thức trên cho thấy cách Adam tính toán bước cập nhật cho tham số θ bằng cách sử dụng $\widehat{m}_t$ và $\widehat{v}_t$. Adam sử dụng bước chuẩn hóa thứ nhất và thứ hai để điều chỉnh giá trị cập nhật dựa trên quá trình tích luỹ gradient và bình phương gradient. Điều này giúp tối ưu hóa việc di chuyển trên bề mặt hàm mất mát và kiểm soát tỷ lệ học động trong quá trình huấn luyện.

Lưu ý rằng các công thức trên chỉ đưa ra một phiên bản cơ bản của Adam. Trong thực tế, có thể có các biến thể và điều chỉnh khác của thuật toán này tùy thuộc vào cài đặt cụ thể và framework sử dụng.

Ưu điểm của Adam:

Hiệu suất cao: Adam thường cho phép huấn luyện mạng nhanh hơn và đạt được hiệu suất tốt hơn so với các thuật toán tối ưu hóa gradient khác như Stochastic Gradient Descent (SGD).

Điều chỉnh tỷ lệ học động tự động: Adam tự động điều chỉnh tỷ lệ học động (learning rate) dựa trên các giá trị đạo hàm của các tham số trong quá trình huấn luyện. Điều này giúp tăng khả năng hội tụ và giảm độ nhạy cảm đến lựa chọn tỷ lệ học động ban đầu.

Điều chỉnh độ nhạy gradient: Adam sử dụng bước chuẩn hóa thứ nhất và thứ hai để điều chỉnh độ nhạy của gradient. Điều này giúp giảm sự dao động và tăng tính ổn định của quá trình huấn luyện.

Hiệu quả đối với các bề mặt không đồng nhất: Adam hoạt động tốt trên các bề mặt hàm không đồng nhất và có độ cong khác nhau trong các hướng khác nhau. Điều này giúp tránh việc rơi vào các điểm tối thiểu cục bộ và tìm được điểm tối ưu toàn cục tốt hơn.

Nhược điểm của Adam:

Yêu cầu bộ nhớ: Adam cần lưu trữ và cập nhật các giá trị mômen (moments) $\widehat{m}_t$ và $\widehat{v}_t$ cho mỗi tham số. Điều này yêu cầu bộ nhớ lớn hơn so với các thuật toán khác như SGD, đặc biệt khi áp dụng cho mạng nơ-ron sâu với nhiều tham số.

Phụ thuộc vào siêu tham số: Adam có hai siêu tham số là β_1 và β_2 cần được điều chỉnh. Lựa chọn không tốt của hai siêu tham số này có thể ảnh hưởng đến hiệu suất của thuật toán.

Khó khăn đối với bộ dữ liệu nhỏ: Adam có thể không hoạt động tốt trên các bộ dữ liệu nhỏ, đặc biệt là khi kích thước của mỗi mini-batch nhỏ hơn. Trong các trường thực tập, nó có thể dẫn đến sự không ổn định và không cân bằng giữa các bước cập nhật.

1.9 AdamW

AdamW là một biến thể của Adam. Ý tưởng của AdamW khá đơn giản: khi thực hiện thuật toán Adam với L2 regularization (chuẩn hóa L2), tác giả loại bỏ phần tiêu biến của trọng số (weight decay) $w_t\theta_t$ khỏi công thức tính gradient hàm mất mát tại thời điểm t :

$$g_t = \nabla f(\theta_t) + w_t \theta_t$$

và thay vào đó, đưa phần giá trị đã được phân tách này vào quá trình cập nhật trọng số:

$$\theta_{t+1,i} = \theta_{t,i} - \eta\left(\frac{1}{\sqrt{\widehat{v}_t + \epsilon}} \cdot \widehat{m}_t + w_{t,i}\theta_{t,i}\right), \forall t$$

Trong đó:

λ là hệ số trọng số suy giảm L2 (L2 weight decay coefficient). Nó được sử dụng để kiểm soát trọng số của mô hình và giảm overfitting bằng cách áp dụng một phạt lớn cho các trọng số lớn.

Các thành phần khác như $\widehat{m}_t$, $\widehat{v}_t$, α và ε giữ nguyên như trong Adam.

Ưu điểm của AdamW:

Giảm overfitting: AdamW kết hợp trọng số suy giảm L2 vào quá trình cập nhật, giúp giảm overfitting bằng cách áp dụng một phạt lớn cho các trọng số lớn. Điều này giúp kiểm soát sự phức tạp của mô hình và cải thiện khả năng khái quát hóa.

Khái quát hóa tốt hơn: Bằng cách áp dụng trọng số suy giảm L2 trực tiếp lên các tham số trong quá trình cập nhật, AdamW giúp giảm phụ thuộc vào các trọng số nhỏ hơn và đồng thời giữ các trọng số quan trọng, giúp cải thiện khả năng khái quát hóa của mô hình.

Đơn giản hóa việc lựa chọn siêu tham số: AdamW giảm sự phụ thuộc vào việc lựa chọn siêu tham số như β_1 và β_2 trong Adam. Thay vào đó, chỉ cần điều chỉnh hệ số trọng số suy giảm L2 λ để kiểm soát độ quan trọng của trọng số.

Nhược điểm của AdamW:

Yêu cầu bộ nhớ: Tương tự như Adam, AdamW cũng yêu cầu lưu trữ và cập nhật các giá trị mômen (moments) và các tham số. Điều này yêu cầu bộ nhớ lớn hơn so với các thuật toán tối ưu hóa gradient khác, đặc biệt là khi áp dụng cho mạng nơ-ron sâu với nhiều tham số.

Phụ thuộc vào siêu tham số: Mặc dù AdamW giảm sự phụ thuộc vào việc lựa chọn siêu tham số so với Adam, nhưng vẫn cần điều chỉnh hệ số trọng số suy giảm L2 λ. Lựa chọn không tốt của λ có thể ảnh hưởng đến hiệu suất của thuật toán.

1.9 So sánh lại ưu nhược điểm của các mô hình

Thuật toán	**Ưu điểm**	**Nhược điểm**
Gradient Descent	- Dễ hiểu và triển khai.	- Có thể mắc phải vấn đề với tốc độ học không tối ưu và mắc kẹt ở điểm cực tiểu cục bộ.
Momentum	- Giúp tăng tốc quá trình hội tụ và tránh rơi vào điểm cực tiểu cục bộ.	- Cần điều chỉnh thêm siêu tham số.
NAG	- Cải thiện Momentum bằng cách tính toán gradient trước khi cập nhật.	- Yêu cầu tính toán gradient thêm.
SGD	- Đơn giản và hiệu quả cho tập dữ liệu lớn.	- Yếu tố ngẫu nhiên có thể khiến quá trình tìm kiếm điểm cực tiểu không ổn định.
SGD with Gradient Clipping	- Giúp kiểm soát độ lớn của gradient và tránh trường hợp	- Cần điều chỉnh thêm giá trị ngưỡng cho gradient.

	gradient phóng đại.	
Adagrad	- Tự động điều chỉnh tỷ lệ học cho từng tham số.	- Tích lũy bình phương gradient có thể dẫn đến tỷ lệ học quá nhỏ.
RMSprop	- Giảm ảnh hưởng của các gradient quá lớn.	- Yêu cầu điều chỉnh thêm siêu tham số.
Adam	- Kết hợp Momentum và Adagrad, kết hợp ưu điểm của cả hai.	- Yêu cầu điều chỉnh thêm siêu tham số.
AdamW	- Cải thiện Adam bằng cách bổ sung trọng số suy giảm L2.	- Yêu cầu điều chỉnh thêm siêu tham số.

Bảng 1: Bảng so sánh lại ưu nhược điểm của các mô hình

II. Continual Learning và Test Production

Trong lĩnh vực Machine Learning (ML) và phát triển hệ thống ML, Continual Learning và Test in Production là hai khái niệm quan trọng nhằm đảm bảo hiệu suất và thích ứng của các mô hình ML trong môi trường thực tế. Trong báo cáo này, chúng ta sẽ tìm hiểu chi tiết về hai khái niệm này và cách chúng đóng vai trò quan trọng trong việc phát triển và triển khai các hệ thống ML.

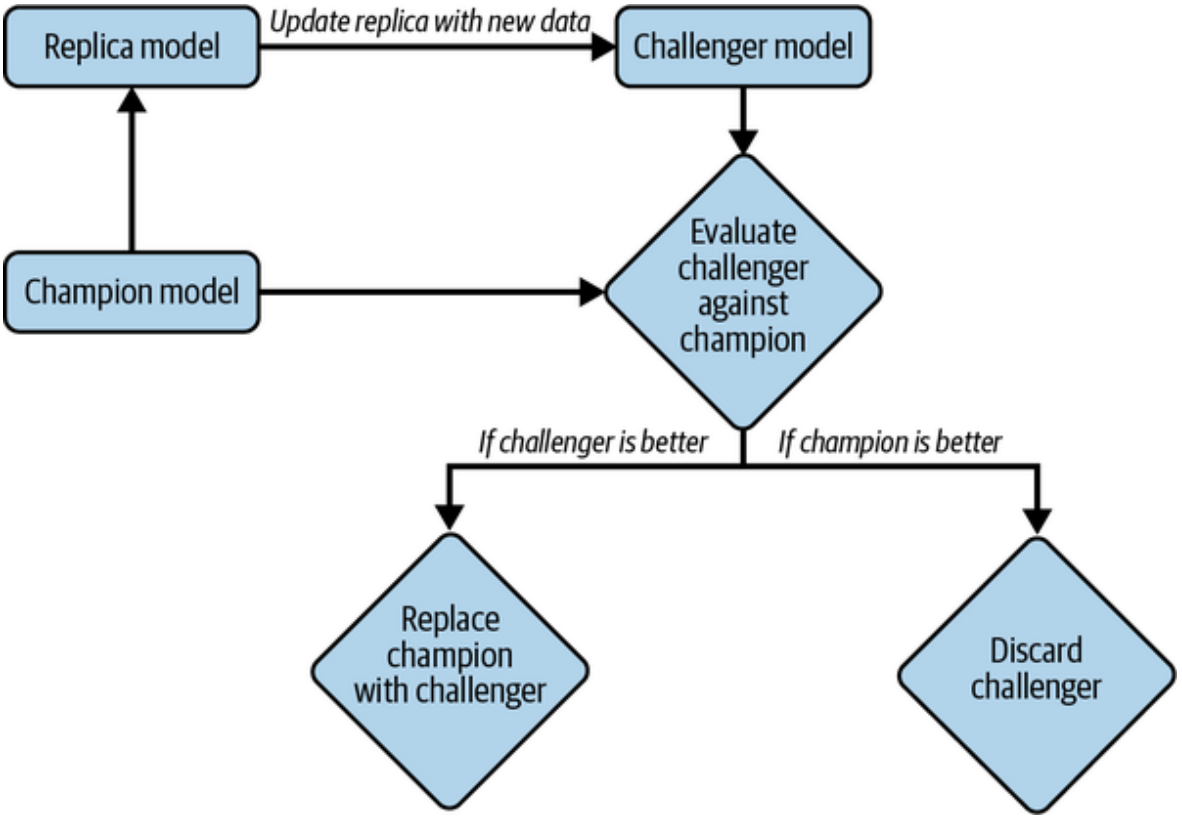
2.1 Continual Learning:.

2.1.1 Định nghĩa và cách tiếp cận Continual Learning

Continual Learning là quá trình liên tục điều chỉnh mô hình ML để thích ứng với sự thay đổi trong phân phối dữ liệu. Mục tiêu của Continual Learning là tự động và an toàn cập nhật mô hình một cách hiệu quả.

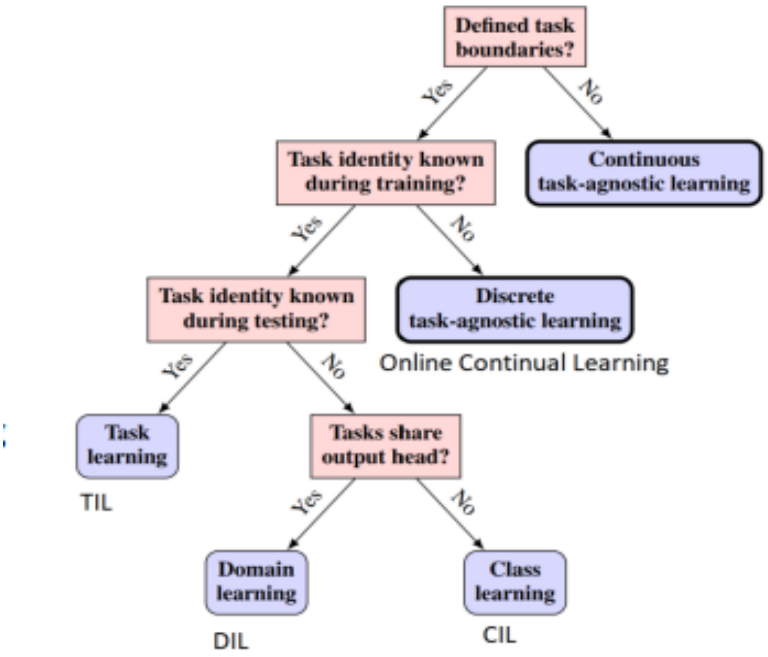
Trong lĩnh vực này, mô hình học máy được đào tạo trên một dòng dữ liệu không ngừng, thường là các tác vụ liên quan hoặc tương tự nhau. Mục tiêu của học liên tục là khả năng học mới mà không quên đi kiến thức đã học trước đó.Nói cách khác, Continual learning quan tâm đến vấn đề quên nghiêm trọng (catastrophic forgetting).Khi huấn luyện trên tasks mới, neural network có xu hướng quên thông

tin đã học được từ những tasks trước.Điều này dẫn đến, làm giảm hiệu suất mô hình trên những tasks cũ.
Hiện tượng này liên quan chặt chẽ đến hiện tượng stability-plasticity dilemma . Nói nói rằng, nếu model quá ổn định thì khó khăn học thông tin mới. Nếu model quá mềm dẻo thì gặp vấn đề quên đi thông tin đã học được.Dưới đây là sơ đồ đơn giản hóa quá trình continual learning:



Hình 7 . Đơn giản hóa cách thức hoạt động của việc học tập liên tục trong sản xuất.

Continual learning được xem xét trong các ML models: SVM , Topic models, decision tree.

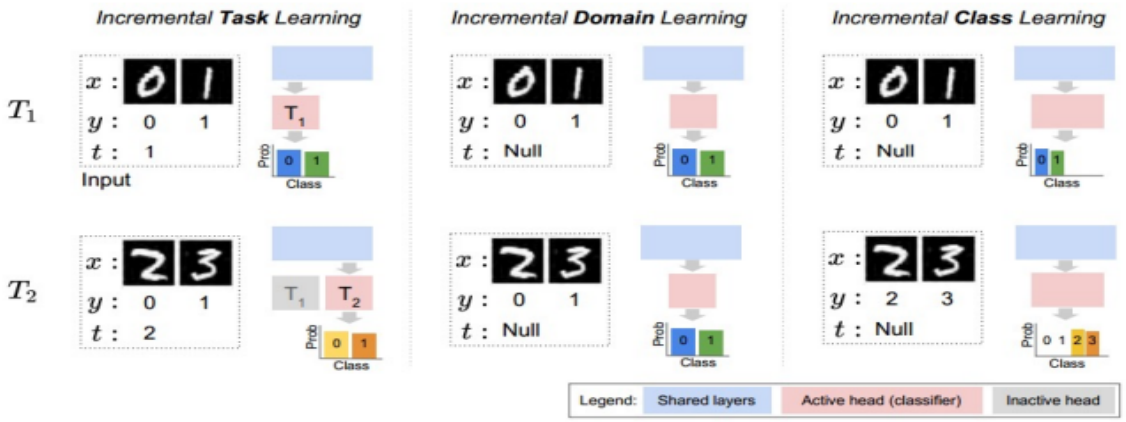


Hình 8: Một số kịch bản chính trong CL

Qua hình trên, ta thấy được Continual learning là học chuỗi các tasks liên tục. Cùng với 3 kịch bản phổ biến là:

- **Task-Incremental Learning (TIL)**
- **Domain-Incremental Learning (DIL)**
- **Class-Incremental Learning (CIL)**

Và, **Online continual learning** là nghiên cứu continual learning trong online setup (dữ liệu đến liên tục, phân phối của dữ liệu thường xuyên thay đổi).



Hình 9: Minh họa cho các kịch bản **TIL, DIL, CIL**

Evaluation Metrics trong CL:

Sử dụng tập hold-out test cho mỗi task trong T tasks. • Huấn luyện lần lượt các task và đánh giá trên tập hold-out test của các tasks: Xây dựng matrix R_{TxT} trong đó $R_{i,j}$ là accuracy của mô hình trên task j khi học xong task i. Có 3 độ đo phổ biến: Average Accuracy (ACC), Backward Transfer (BWT), Forward Transfer (FWT).

- **ACC=** $\frac{1}{T}\sum_{i=1}^{T} R_{T,i}$
- **BWT=** $\frac{1}{T-1}\sum_{i=1}^{T} (R_{T,i} - R_{i,i})$
- **FWT=** $\frac{1}{T-1}\sum_{i=1}^{T} R_{i-1,i}$

Các cách tiếp cận trong CL:

Ba cách tiếp cận phổ biến:

• Prior/Regularization-based approach:

- Bổ sung đại lượng regularization hoặc distillation loss vào hàm loss hiện tại để giữ tham số mô hình hiện tại "gần" với tham số mô hình ở task trước:

$$\mathcal{L}_k(\theta) = \mathcal{L}_k(\theta) + \lambda(\theta - \theta^{t-1})^T \Omega^{t-1}(\theta - \theta^{t-1})$$

trong đó, với θ là tham số mô hình (weights), $\mathcal{L}_k$ là hàm loss cho task , Ω^{t-1} mã hóa độ quan trọng của weights.

Nhiều cách khác nhau mã hóa độ quan trọng:

- ❖ Elastic Weight Consolidation (EWC) sử dụng Fisher Information
- ❖ Synaptic Intelligence (SI) dựa trên giảm của hàm loss
- ❖ Memory Aware Synapses (MAS) dựa trên biến đổi của đầu ra (output)

- Variational Continual Learning (VCL) sử dụng cách tiếp cận Bayes:

$$P(\theta|(D_1, D_2,..., D_t) \propto P(D_t|\theta)P(\theta, D_1, D_2,.... D_{t-1})$$

Với D_t là dữ liệu huấn luyện của task t.

Tri thức học được ở những task trước đóng vai trò là tiên nghiệm (prior) cho task hiện tại:

$$P(\theta, D_1, D_2,.... D_{t-1}) = q_{t-1}(\theta)$$

Sử dụng online variational inference:

$$E_{q(\theta)} logP(D_t|\theta) - KL(q(\theta))||q_{t-1}(\theta))$$

với p(θ) là xấp xỉ phân phối hậu nghiệm ở task hiện tại

• Memory-based approach:

Sử dụng bộ nhớ nhỏ (memory buffer) để lưu lại data từ các tasks trước và huấn luyện lại cùng với task hiện tại (rehearsal).

Nhiều chiến lược lựa chọn data đặc trưng được đề xuất:

- ❖ Chọn data đại diện dựa theo lớp (herding algorithm).
- ❖ Chọn data "khó": Nằm gần biên phân loại (decision boundary) hoặc gây ra loss lớn.
- ❖ Sử dụng các chiến lược bi-level optimization.

Xây dựng các mô hình generative models để sinh lại dữ liệu cũ hoặc sinh lại biểu diễn của dữ liệu cũ. Gradient Episodic Memory (GEM) and A-GEM lưu dữ liệu để ngăn cập nhật weight theo hướng gradient làm giảm loss trên dữ liệu cũ.

• Architecture-based approach:

Học một phần kiến trúc (sub-network) cho mỗi task và lưu sub-network. Đóng băng phần kiến trúc này và mở rộng mạng để học task mới.

Lặp lại 2 bước:

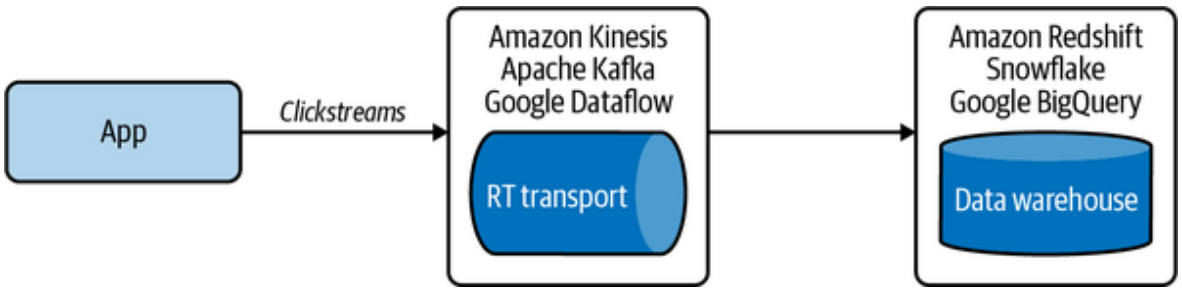
- ❖ Network expansion
- ❖ Sparse learning

Với TIL, tiếp cận dựa trên kiến trúc không bị quên task cũ. Với DIL và CIL, cần bước xác định task boundary khi inference.

2.1.2 Thách thức

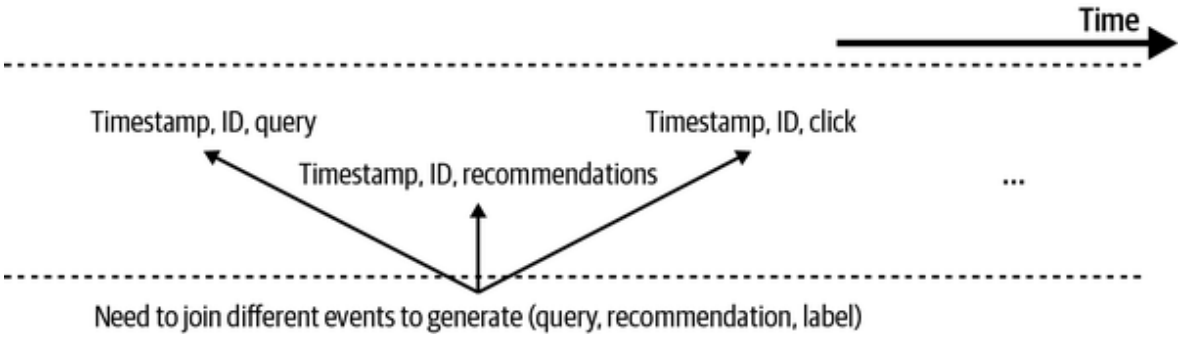
Mặc dù học tập liên tục được sử dụng nhiều đã áp dụng nó rất thành công nhưng học tập liên tục vẫn còn nhiều thách thức. Trong phần này, chúng ta sẽ thảo luận về ba thách thức chính: truy cập, đánh giá và thuật toán dữ liệu mới.

a. Thách thức có được dữ liệu mới: Nếu bạn muốn cập nhật mô hình của mình mỗi giờ, bạn cần dữ liệu mới mỗi giờ. Hiện nay,người ta lấy dữ liệu đào tạo mới từ kho dữ liệu của họ. Tốc độ bạn có thể lấy dữ liệu từ kho dữ liệu của mình phụ thuộc vào tốc độ dữ liệu này được gửi vào kho dữ liệu của bạn. Tốc độ có thể chậm, đặc biệt nếu dữ liệu đến từ nhiều nguồn. Một cách khác là cho phép kéo dữ liệu trước khi nó được gửi vào kho dữ liệu, ví dụ: trực tiếp từ các phương tiện truyền tải thời gian thực như Kafka và Kinesis vận chuyển dữ liệu từ ứng dụng đến kho dữ liệu, như trong hình sau:



Hình 10: Việc kéo dữ liệu trực tiếp từ quá trình vận chuyển theo thời gian thực trước khi được gửi vào kho dữ liệu có thể cho phép bạn truy cập dữ liệu mới hơn

Có thể lấy dữ liệu mới là không đủ. Nếu mô hình của bạn cần dữ liệu được gắn nhãn để cập nhật, như hầu hết các mô hình ngày nay đều làm, thì dữ liệu này cũng cần được gắn nhãn. Trong nhiều ứng dụng, tốc độ cập nhật mô hình bị tắc nghẽn bởi tốc độ dán nhãn dữ liệu.Những ứng cử viên tốt nhất cho việc học tập liên tục là những nhiệm vụ mà bạn có thể nhận được những nhãn hiệu tự nhiên với các vòng phản hồi ngắn. Ví dụ về các nhiệm vụ này là định giá động (dựa trên nhu cầu và tính sẵn có ước tính), ước tính thời gian đến, dự đoán giá cổ phiếu, dự đoán nhấp qua quảng cáo và hệ thống đề xuất cho nội dung trực tuyến như tweet, bài hát, video ngắn, bài viết, v.v .Tuy nhiên, những nhãn tự nhiên này thường không được tạo ra dưới dạng nhãn mà là các hoạt động hành vi cần được trích xuất thành nhãn. Hãy xem qua một ví dụ để làm rõ điều này. Nếu bạn điều hành một trang web thương mại điện tử, ứng dụng của bạn có thể đăng ký vào lúc 10:33 tối, người dùng A nhấp vào sản phẩm có ID là 32345. Hệ thống của bạn cần xem lại nhật ký để xem ID sản phẩm này có từng được đề xuất cho điều này hay không người dùng và nếu có thì truy vấn nào đã nhắc đề xuất này để hệ thống của bạn có thể khớp truy vấn này với đề xuất này và gắn nhãn đề xuất này là đề xuất tốt, như minh họa trong:



Hình 11:Đơn giản hóa quy trình trích xuất nhãn từ phản hồi của người dùng

b. Thách thức đánh giá

Thách thức lớn nhất của việc liên tục việc học không phải là viết một hàm để liên tục cập nhật mô hình của bạn, bạn có thể làm điều đó bằng cách viết một tập lệnh.Thách thức lớn nhất là đảm bảo rằng bản cập nhật này đủ tốt để triển khai. Rủi ro về những thất bại sẽ tăng lên khi không ngừng học hỏi.

Đầu tiên, bạn cập nhật mô hình của mình càng thường xuyên thì càng có nhiều khả năng xảy ra lỗi cập nhật.Thứ hai, việc học tập liên tục sẽ làm cho mô hình của bạn trở nên tốt hơn dễ bị thao túng phối hợp và tấn công đối nghịch hơn. Vì các mô hình của bạn học trực tuyến từ dữ liệu trong thế giới thực nên người dùng dễ dàng nhập dữ liệu độc hại hơn để lừa các mô hình học những điều sai.

Khi thiết kế quy trình đánh giá để học hỏi liên tục, hãy nhớ rằng việc đánh giá cần có thời gian, đây có thể là một trở ngại khác đối với tần suất cập nhật mô hình. Ví dụ: một công ty thanh toán trực tuyến lớn mà tôi từng làm việc có hệ thống ML để phát hiện các giao dịch gian lận. Các hình thức lừa đảo thay đổi nhanh chóng nên họ muốn cập nhật hệ thống của mình nhanh chóng để thích ứng với các hình thức

thay đổi. Họ không thể triển khai mô hình mới trước khi nó được thử nghiệm A/B so với mô hình hiện tại. Tuy nhiên, do tính chất mất cân bằng của nhiệm vụ, hầu hết các giao dịch không phải là gian lận, họ phải mất khoảng hai tuần để xem đủ số giao dịch gian lận để có thể đánh giá chính xác mô hình nào tốt hơn. Vì vậy, họ chỉ có thể cập nhật hệ thống của mình mỗi hai tuần.

c. Thách thức thuật toán

So với thách thức dữ liệu mới và đánh giá, đây là một thách thức dễ dành hơn vì nó chỉ ảnh hưởng đến một số thuật toán nhất định và tần suất đào tạo nhất định. Nói chính xác thì nó chỉ ảnh hưởng đến các mô hình matrix-based và tree-based cần được cập nhật rất nhanh (ví dụ: hàng giờ).

Để minh họa điểm này, hãy xem xét hai mô hình khác nhau: mạng nơ-ron và mô hình matrix-based, chẳng hạn như mô hình collaborative filtering. Mô hình collaborative filtering sử dụng ma trận mục người dùng và kỹ thuật giảm kích thước.

Bạn có thể cập nhật mô hình nơ-ron với một dữ liệu có kích thước bất kỳ. Bạn thậm chí có thể thực hiện bước cập nhật chỉ với một mẫu dữ liệu. Tuy nhiên, nếu muốn cập nhật mô hình collaborative filtering, trước tiên bạn cần sử dụng toàn bộ tập dữ liệu để xây dựng ma trận mục người dùng trước khi thực hiện giảm kích thước trên đó. Tất nhiên, bạn có thể áp dụng giảm kích thước cho ma trận của mình mỗi khi cập nhật ma trận với mẫu dữ liệu mới, nhưng nếu ma trận của bạn lớn thì bước giảm kích thước sẽ quá chậm và tốn kém để thực hiện thường xuyên. Do đó,

mô hình này ít phù hợp hơn cho việc học với tập dữ liệu một phần so với mô hình mạng nơ-ron trước đó.

2.1.3 Bốn giai đoạn Continual Learning:

Giai đoạn 1: Huấn luyện thủ công, không lưu trạng thái (Manual, stateless retraining): Trong giai đoạn này, mô hình máy học được huấn luyện bằng cách từ đầu (stateless), tức là không lưu trạng thái trước đó. Mỗi lần cập nhật mô hình, toàn bộ quá trình huấn luyện được thực hiện lại từ đầu.

Giai đoạn 2: Huấn luyện tự động (Automated retraining): Ở giai đoạn này, quá trình huấn luyện mô hình được tự động hóa. Hệ thống sẽ tự động thu thập dữ liệu mới, huấn luyện và triển khai các mô hình cập nhật. Việc này giúp giảm công sức và thời gian cần thiết cho việc cập nhật mô hình.

Giai đoạn 3: Huấn luyện tự động, lưu trạng thái (Automated, stateful training): Tại giai đoạn này, mô hình máy học được huấn luyện tự động và cũng cho phép lưu trạng thái (stateful). Thay vì huấn luyện mô hình từ đầu, mô hình sẽ tiếp tục được huấn luyện trên dữ liệu mới để cải thiện hiệu suất. Việc này giúp giảm yêu cầu về dữ liệu và tăng tốc quá trình huấn luyện.

Giai đoạn 4: Học liên tục (Continual learning): Giai đoạn cuối cùng tập trung vào việc thực hiện học liên tục trong hệ thống máy học. Quá trình này bao gồm cả việc cập nhật mô hình và huấn luyện liên tục để thích ứng với sự thay đổi của môi trường và dữ liệu. Mục tiêu là duy trì và cải thiện hiệu suất mô hình máy học theo thời gian.

2.1.4 Model iteration versus data iteration

Mô hình lặp lại (Model iteration): Trong quá trình mô hình lặp lại, chúng ta tập trung vào việc cải tiến và thay đổi mô hình máy học. Các thay đổi này có thể bao gồm việc thay đổi kiến trúc mô hình, thêm hoặc loại bỏ các thành phần trong mô hình, điều chỉnh siêu tham số, hoặc sử dụng các kỹ thuật tối

ưu hóa khác nhau. Mô hình lặp lại giúp chúng ta cải thiện hiệu suất và khả năng thích ứng của mô hình với dữ liệu mới.

Dữ liệu lặp lại (Data iteration): Trong quá trình dữ liệu lặp lại, chúng ta tập trung vào việc thu thập và sử dụng dữ liệu mới để huấn luyện và cập nhật mô hình. Điều này bao gồm việc thu thập dữ liệu mới từ môi trường hoặc nguồn dữ liệu, xử lý và tiền xử lý dữ liệu, và sau đó sử dụng dữ liệu này để huấn luyện lại mô hình. Quá trình này giúp chúng ta cải thiện khả năng dự đoán và thích ứng của mô hình với sự thay đổi trong dữ liệu.

2.2 Test Production

Test Production đề cập đến quá trình triển khai mô hình học máy vào một môi trường thực tế và đánh giá hiệu suất của nó. Vì sự phân bổ dữ liệu thay đổi nên việc một mô hình hoạt động tốt trên dữ liệu từ giờ trước không có nghĩa là nó sẽ tiếp tục hoạt động tốt trên dữ liệu trong tương lai. Cách duy nhất để biết liệu một mô hình có hoạt động tốt hay không là triển khai nó. Cái nhìn sâu vấn đề này đã dẫn đến một khái niệm rất cần thiết: **Test Production.** Tuy nhiên, việc **Test Production** không thật sự khó khăn. Có các kỹ thuật giúp bạn đánh giá (hầu hết) các mô hình của mình một cách an toàn. Trong phần này, chúng tôi sẽ đề cập đến các kỹ thuật sau: shadow deployment, A/B testing, canary analysis, interleaving experiments, and bandits.

2.2.1 Shadow deployment

Triển khai Shadow (Triển khai Bóng) có thể là cách an toàn nhất để triển khai mô hình hoặc bất kỳ cập nhật phần mềm nào. Shadow Deployment hoạt động như sau:

Triển khai mô hình ứng cử viên song song với mô hình hiện tại.Đối với mỗi yêu cầu đến, định tuyến yêu cầu đó đến cả hai mô hình để thực hiện dự đoán, nhưng chỉ phục vụ dự đoán của mô hình hiện tại cho người dùng.

Ghi lại các dự đoán từ mô hình mới để phân tích.Chỉ khi bạn đã xác định rằng các dự đoán từ mô hình mới là đáng tin cậy, bạn mới thay thế mô hình hiện tại bằng mô hình mới.

Bằng việc không phục vụ dự đoán từ mô hình mới cho người dùng cho đến khi đã đảm bảo rằng dự đoán của mô hình đạt yêu cầu, rủi ro mô hình mới có hành vi không mong muốn là thấp, ít nhất là không cao hơn mô hình hiện tại. Tuy nhiên, phương pháp này không phù hợp trong mọi trường hợp vì nó tốn kém. Nó làm tăng gấp đôi số dự đoán mà hệ thống của bạn phải tạo ra, điều này thường đồng nghĩa với việc tăng gấp đôi chi phí tính toán dự đoán.

Triển khai Shadow được coi là một phương pháp giá trị khi đảm bảo tính đáng tin cậy và hiệu suất của một mô hình mới trước khi triển khai hoàn toàn cho người dùng.

2.2.2 A/B Testing

A/B testing (kiểm tra A/B) là một cách để so sánh hai biến thể của một đối tượng, thông thường bằng cách kiểm tra phản hồi đối với hai biến thể này và xác định biến thể nào hiệu quả hơn trong hai biến thể đó. Trong trường hợp của chúng ta, mô hình hiện tại là một biến thể và mô hình ứng cử viên (mô hình đã được cập nhật gần đây) là biến thể khác. Chúng ta sẽ sử dụng kiểm tra A/B để xác định mô hình nào tốt hơn dựa trên một số chỉ số đã được xác định trước.
Nó bao gồm các bước cơ bản:
1. Triển khai mô hình ứng cử viên cùng với mô hình hiện tại.
2. Một phần trăm định tuyến đến mô hình mới để thực hiện dự đoán; phần còn lại được định tuyến đến mô hình hiện tại để thực hiện dự đoán. Thông thường, cả hai biến thể đều phục vụ lưu lượng dự đoán cùng một lúc. Tuy nhiên, có những trường hợp mà dự đoán của một mô hình có thể ảnh hưởng đến dự đoán của mô hình khác.
3. Theo dõi và phân tích các dự đoán và phản hồi từ người dùng, nếu có, từ cả hai mô hình để xác định sự khác biệt về hiệu suất giữa hai mô hình có ý nghĩa thống kê.

Để thực hiện kiểm tra A/B một cách chính xác, bạn cần làm nhiều việc .
Đầu tiên, kiểm tra A/B bao gồm một thí nghiệm ngẫu nhiên: lưu lượng được định tuyến đến mỗi mô hình phải thực sự ngẫu nhiên. Nếu không, kết quả kiểm tra sẽ không hợp lệ. Ví dụ, nếu có sai lệch lựa chọn trong cách lưu lượng được định tuyến đến hai mô hình, chẳng hạn như người dùng được tiếp xúc với mô hình A thường sử dụng điện thoại di động còn người dùng tiếp xúc với mô hình B thường sử dụng máy tính để bàn, thì nếu mô hình A có độ chính xác tốt hơn mô hình B, chúng ta không thể biết liệu điều đó đến từ việc "sử dụng điện thoại di động" hay không.

Thứ hai, kiểm tra A/B của bạn nên được thực hiện trên một số lượng mẫu đủ lớn để có đủ độ tin cậy về kết quả. Cách tính số lượng mẫu cần thiết cho kiểm tra A/B là một câu hỏi đơn giản nhưng có một câu trả lời phức tạp.

Thường, trong production, bạn không chỉ có một ứng cử viên mà còn có nhiều mô hình ứng cử viên. Có thể thực hiện kiểm tra A/B với nhiều hơn hai biến thể, điều này có nghĩa là chúng ta có thể có kiểm tra A/B/C hoặc thậm chí kiểm tra A/B/C/D.

2.2.3 Canary Release

Canary release là một kỹ thuật nhằm giảm rủi ro khi triển khai phiên bản phần mềm mới vào môi trường production bằng cách mở rộng thay đổi một cách từ từ cho một tập con nhỏ người dùng trước khi triển khai cho toàn bộ hệ thống và làm cho phiên bản mới có sẵn cho tất cả mọi người. Trong ngữ cảnh triển khai học máy, canary release hoạt động như sau:

1. Triển khai mô hình ứng cử viên (canary) cùng với mô hình hiện tại. Mô hình ứng cử viên được gọi là canary.
2. Một phần lưu lượng được định tuyến đến mô hình ứng cử viên.
3. Nếu hiệu suất của canary đạt yêu cầu, tăng lưu lượng đến mô hình ứng cử viên. Nếu không, hủy bỏ canary và định tuyến toàn bộ lưu lượng về mô hình hiện tại.
4. Dừng lại khi canary phục vụ toàn bộ lưu lượng (mô hình ứng cử viên đã thay thế mô hình hiện tại) hoặc khi canary bị hủy bỏ.

Hiệu suất của mô hình ứng cử viên được đo lường so với hiệu suất của mô hình hiện tại dựa trên các chỉ số mà bạn quan tâm. Nếu các chỉ số quan trọng của mô hình ứng cử viên suy giảm đáng kể, canary sẽ bị hủy bỏ và toàn bộ lưu lượng sẽ được định tuyến về mô hình hiện tại.

Canary release có thể được sử dụng để thực hiện kiểm tra A/B do sự tương đồng trong cách triển khai. Tuy nhiên, bạn có thể thực hiện phân tích canary mà không cần sử dụng kiểm tra A/B. Ví dụ, bạn không cần phải ngẫu nhiên hóa lưu lượng để định tuyến đến mỗi mô hình. Một kịch bản hợp lý là bạn triển khai mô hình ứng cử viên vào thị trường ít quan trọng hơn trước khi triển khai cho tất cả mọi người.

2.2.4 Bandit

Cho những người không quen thuộc, thuật toán bandit bắt nguồn từ hoạt động đánh bạc. Một sòng bạc có nhiều máy đánh bạc với tỷ lệ trả thưởng khác nhau. Một máy đánh bạc cũng được gọi là một "tên trộm một cánh tay", do đó được gọi là bandit. Bạn không biết máy đánh bạc nào cho tỷ lệ trả thưởng cao nhất. Bạn có thể tiến

hành thử nghiệm để tìm ra máy đánh bạc tốt nhất trong quá trình tối đa hóa số tiền nhận được. Các thuật toán bandit đa cánh cho phép bạn cân bằng giữa khai thác (chọn máy đánh bạc đã trả nhiều nhất trong quá khứ) và khám phá (chọn các máy đánh bạc khác có thể trả thưởng cao hơn).

Hiện nay, phương pháp tiêu chuẩn để kiểm tra các mô hình trong sản xuất là thử nghiệm A/B. Với thử nghiệm A/B, bạn ngẫu nhiên chuyển hướng lưu lượng truy cập đến mỗi mô hình để dự đoán và đo lường cuối cùng xem mô hình nào hoạt động tốt hơn. Thử nghiệm A/B không có trạng thái: bạn có thể chuyển hướng lưu lượng truy cập đến mỗi mô hình mà không cần biết về hiệu suất hiện tại của chúng. Bạn có thể thực hiện thử nghiệm A/B ngay cả với dự đoán theo lô.

Khi bạn có nhiều mô hình để đánh giá, mỗi mô hình có thể được coi là một máy đánh bạc có số tiền nhận được (tức độ chính xác dự đoán) mà bạn không biết. Bandit cho phép bạn xác định cách chuyển hướng lưu lượng truy cập đến mỗi mô hình để dự đoán và xác định mô hình tốt nhất trong quá trình tối đa hóa độ chính xác dự đoán cho người dùng của bạn. Bandit có trạng thái: trước khi chuyển hướng yêu cầu sang một mô hình, bạn cần tính toán hiệu suất hiện tại của tất cả các mô hình. Điều này yêu cầu ba yếu tố:

1. Mô hình của bạn phải có khả năng thực hiện dự đoán trực tuyến.
2. Ưu tiên các vòng lặp phản hồi ngắn: bạn cần phản hồi về việc dự đoán có tốt hay không. Điều này thường đúng đối với các nhiệm vụ mà nhãn có thể được xác định từ phản hồi của người dùng, chẳng hạn như trong các khuyến nghị - nếu người dùng nhấp vào một khuyến nghị, nó được cho là tốt. Nếu vòng lặp phản hồi ngắn, bạn có thể nhanh chóng cập nhật số tiền nhận được của mỗi mô hình.
3. Một cơ chế để thu thập phản hồi, tính toán và theo dõi hiệu suất của mỗi mô hình, và chuyển tiếp các yêu cầu dự đoán đến các mô hình khác nhau dựa trên hiệu suất hiện tại của chúng.

Bandit đã được nghiên cứu kỹ lưỡng trong giới học thuật và đã được chứng minh là cung cấp hiệu suất sử dụng dữ liệu cao hơn so với thử nghiệm A/B (trong nhiều trường hợp, bandit thậm chí là tối ưu). Bandit yêu cầu ít dữ liệu hơn để xác định mô hình tốt nhất và đồng thời giảm thiểu chi phí cơ hội bằng cách chuyển hướng lưu lượng truy cập đến mô hình tốt hơn nhanh chóng hơn.Tuy nhiên, bandit khá khó triển khai hơn so với thử nghiệm A/B vì nó yêu cầu tính toán và theo dõi số tiền nhận được của các mô hình.

TÀI LIỆU THAM KHẢO

[1] Trực, T. T. (2020, October 17). Optimizer- Hiểu sâu về các thuật toán tối ưu (GD, SGD, Adam,..)

[2] Géron, A. (2019). Designing Machine Learning Systems. Sebastopol, CA: O'Reilly Media.