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

BÁO CÁO CUỐI KỲ MÔN HỌC MÁY

Người hướng dẫn: **PSG. TS. LÊ ANH CƯỜNG**

Người thực hiện: **ĐOÀN TRẦN QUỐC TOÀN – 52100492**

Lớp **: 21050401**

Khoá **: 25**

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

BÁO CÁO CUỐI KỲ MÔN HỌC MÁY

Người hướng dẫn: **PSG. TS. LÊ ANH CƯỜNG**

Người thực hiện: **ĐOÀN TRẦN QUỐC TOÀN – 52100492**

Lớp **: 21050401**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành đến thầy về những kiến thức quý báu và sự hướng dẫn tận tâm trong môn học máy bài giữa kỳ. Thầy đã giúp em hiểu rõ hơn về chủ đề này và trang bị cho em những kỹ năng quan trọng.Em hết sức biết ơn công lao của thầy và cam kết nỗ lực hết mình để hoàn thành tốt môn học. Mong rằng thầy sẽ tiếp tục hỗ trợ và khuyến khích em trong quá trình học tập.

Một lần nữa, em xin bày tỏ lòng biết ơn sâu sắc đến thầy.

Trân trọng,

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

Tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của TS Nguyễn Văn A;. 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.

Ngoài ra, trong đồ án 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.

Nếu phát hiện có bất kỳ sự gian lận nào 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 tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm

Tác giả

(ký tên và ghi rõ họ tên)

Đoàn Trần Quốc Toàn

Nguyễn Quang Lợi

Đỗ Nhật Khả Vy

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)

TÓM TẮT

Bài nghiên cứu của tôi tập trung vào hai chủ đề quan trọng trong lĩnh vực học máy: phương pháp Optimizer trong huấn luyện mô hình và Continual Learning cùng Test Production khi áp dụng vào giải quyết bài toán cụ thể.

1) Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy:Trong nghiên cứu của mình, tôi đã tiến hành nghiên cứu chi tiết và so sánh các phương pháp Optimizer phổ biến như Gradient Descent, Stochastic Gradient Descent (SGD), Adam, RMSprop, và AdaGrad. Tôi đã đánh giá hiệu suất của từng phương pháp này dựa trên khả năng hội tụ, tốc độ huấn luyện, và khả năng tránh trượt qua điểm cực tiểu cục bộ.

2) Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán cụ thể:Tôi đã nghiên cứu sâu về Continual Learning - việc học từ dữ liệu liên tục mà không quên đi kiến thức đã học. Đồng thời, tôi cũng tập trung vào Test Production, quá trình tạo ra tập dữ liệu kiểm thử phù hợp để đánh giá hiệu suất của mô hình trong môi trường thực tế. Áp dụng kiến thức này vào việc giải quyết một bài toán cụ thể, tôi đã phát triển một giải pháp học máy linh hoạt, có khả năng học tập liên tục và cung cấp các bộ dữ liệu kiểm thử thích hợp.

Kết quả của nghiên cứu đã cho thấy rằng việc lựa chọn phương pháp Optimizer phù hợp có thể cải thiện đáng kể hiệu suất của mô hình. Đồng thời, việc áp dụng Continual Learning và Test Production đã giúp mô hình của tôi đạt được kết quả tốt hơn khi đối mặt với dữ liệu mới và không chắc chắn.

Tóm lại, nghiên cứu này không chỉ cung cấp cái nhìn sâu rộng về các phương pháp Optimizer và Continual Learning mà còn đề xuất một giải pháp học máy linh hoạt và hiệu quả trong thực tế khi áp dụng vào các bài toán học máy cụ thể

MỤC LỤC

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

CHƯƠNG 1 – TÌM HIỂU, SO SÁNH CÁC PHƯƠNG PHÁP OPTIMIZER TRONG HUẤN LUYỆN MÔ HÌNH HỌC MÁY

1.1 Optimizer là gì?

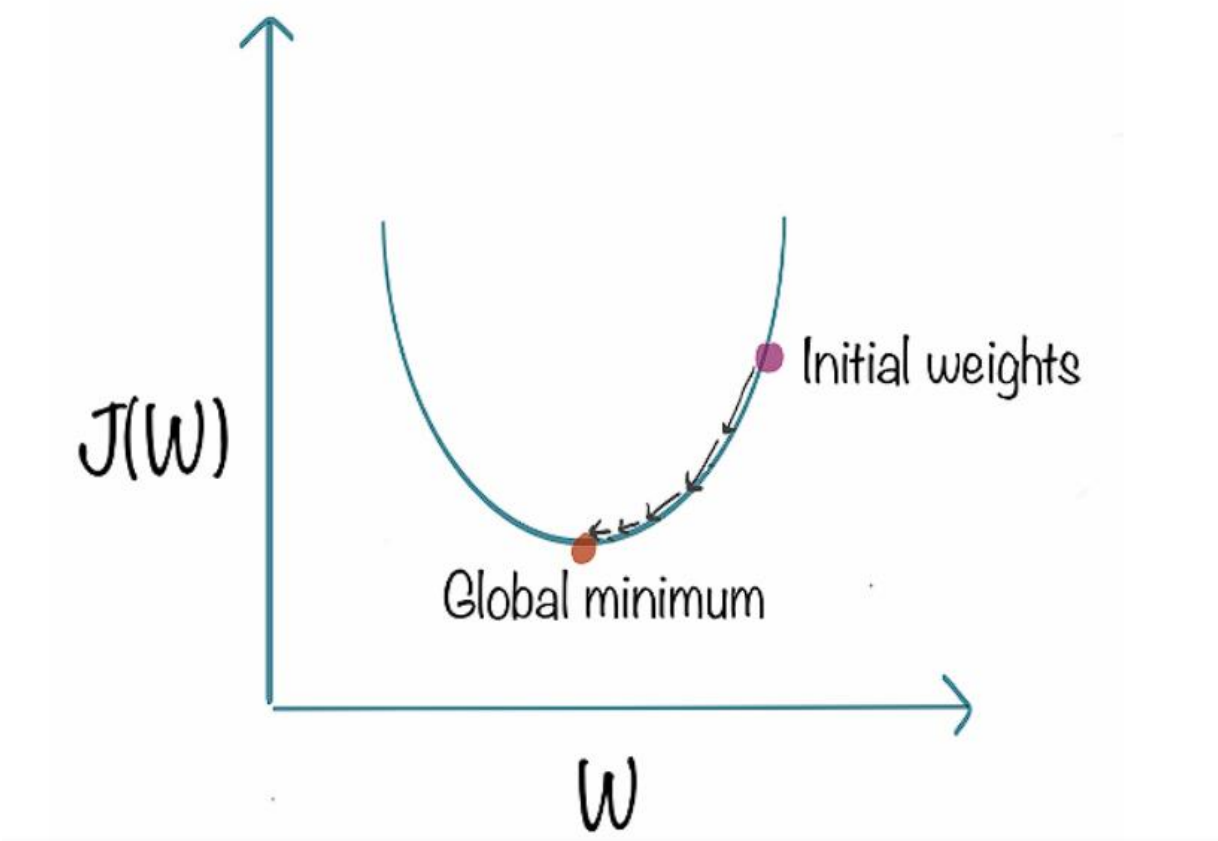
- Thuật toán Optimizer là cơ sở để xây dựng mô hình neural network. Mục đích của của chúng là “học” được các features (hay fattern) của dữ liệu đầu vào. Việc "học" trong mô hình neural network đề cập đến việc điều chỉnh các weights và bias sao cho mô hình có thể dự đoán chính xác hơn từ dữ liệu huấn luyện.
- Trước khi đi sâu ta sẽ tìm hiểu xem grandient là gì? Đơn giản, hãy tưởng tượng bạn đang cầm một quả bóng đang nằm ở đỉnh của một cái bát. Khi bạn buông bóng, nó đi theo hướng dốc nhất và cuối cùng dừng lại ở đáy của cái bát. Gradient cung cấp hướng dốc nhất cho quả bóng để đạt đến điểm cực tiểu cục bộ, tức là đáy của cái bát.
- Thay vì chọn ngẫu nhiên các giá trị, thuật toán sẽ tìm các giá trị tối ưu nhất dự trên quy trình học có cấu trúc hơn. Quy trình này thường được dựa trên việc tính Grandient của Loss function liên quan đến weight và bias. Grandient cho biết hướng và độ lớn mà loss function thay đổi khi trọng số thay đổi.
- Thuật toán Optimizer sử dụng grandient để điều chỉnh các trọng số theo hướng giảm mất mát thông qua việc lặp lại quá trình này trên các batch dữ liệu, Optimizer sẽ tiến gần đến điểm tối ưu hoặc điểm cực tiểu của Loss function. Bằng cách điều chỉnh weight và bias theo Grandient, mô hình có thể học các đặc trưng, patterns từ dữ liệu và tối ưu hoá hiệu suất.
- Vấn đề ở đây là lựa chọn Optimizer phù hợp hợp với các bài toán cụ thể. Một số Optimizer có thể hội tụ nhanh tránh các vùng cực tiểu, hoặc tối ưu hoá với

các dữ liệu dầu vào lớn mà không cần phải sử dụng nhiều tài nguyên tính toán phức tạp.

1.2 Các thuật toán Optimizer tối ưu:

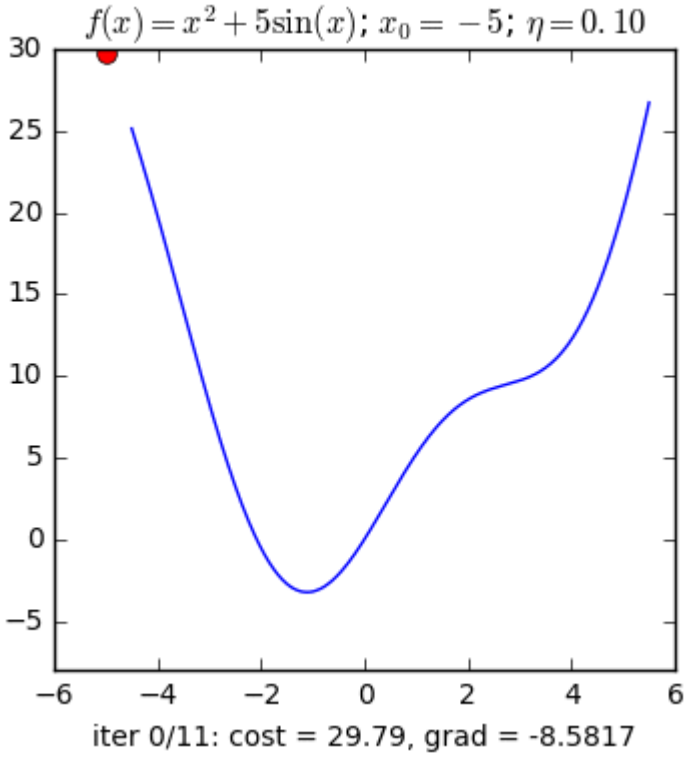
1.2.1 Grandient Descent (GD):

- Gradient descent có thể được xem là một thuật toán được yêu thích nhất trong các thuật toán Optimizers. Thuật toán này sử dụng phép tính vi phần để liên tục điều chỉnh giá trị nhằm đặt đến giá trị của điểm cực tiểu cục bộ.
- Công thức trên mô tả cách tính đạo hàm của hàm mất mát:
    - $x_{new} = x - alpha * f'(x)$
    - Trong đó alpha là bước nhảy biểu thị khoảng các di chuyển của thuật toán theo mỗi đạo hàm theo mỗi vòng lặp.
- Thuật toán Grandient hoạt động như sau:
    - Bắt đầu với các hệ số ban đầu, đánh giá các giá trị tương ứng, và tìm kiếm các giá trị thấp hơn với các giá trị hiện tại.
    - Tiến dần đến các trọng số thấp hơn và cập nhật dần giá trị của các hệ số.
    - Qúa trình này được lặp lại liên tục đến khi tìm được giá trị cực tiểu cục bộ.



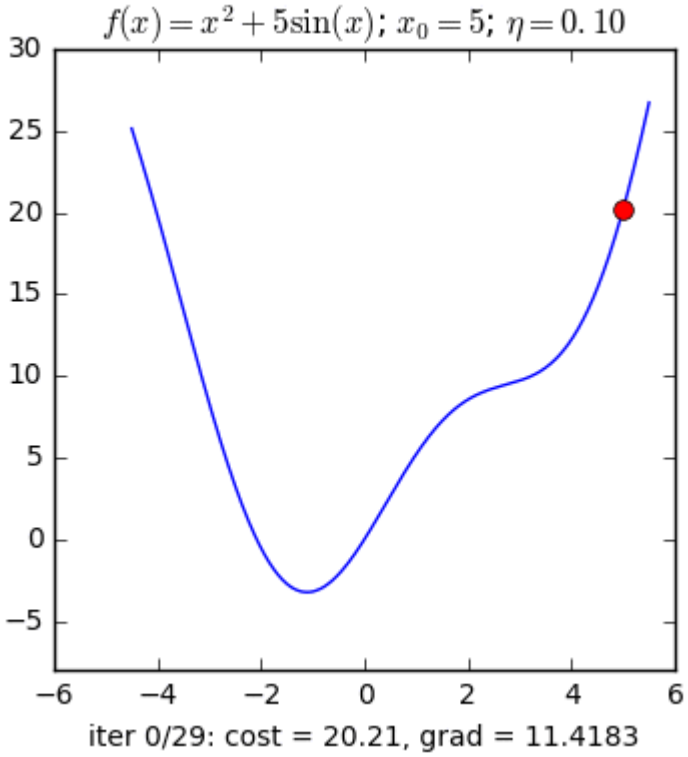
Hình 1 Hình ảnh mô tả các hoạt động của grandient descent

1.2.1.1 Gradient descent cho hàm 1 biến:

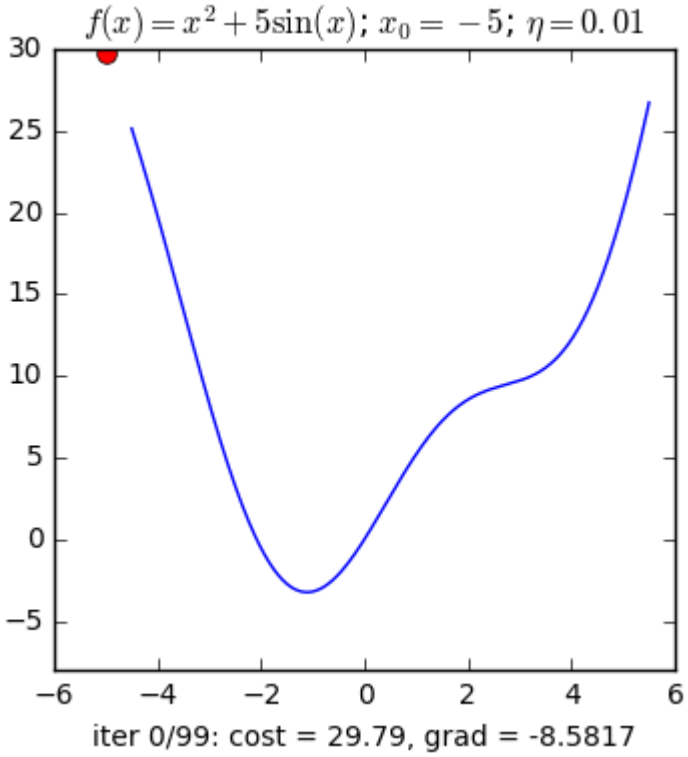


Hình 2 Gradient descent cho hàm một biến $f(x) = x^2 + 5\sin(x)\,; x_0 = -5; n = 0.10$

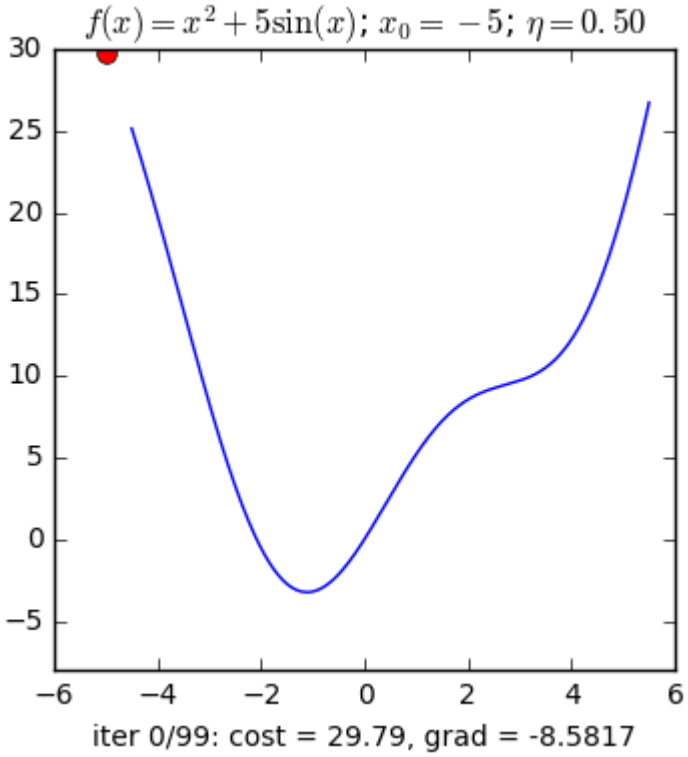
- Bài toán này yêu cầu tìm giá trị của hàm số f(x) = x^2 + 5sin(x) tại điểm x_0 = -5 với bước nhảy n = 0.10
    - Bằng cách áp dụng công thức $x_{new} = x - n * f'(x)$
    - Trong đó $f'(x) = 2x + 5\cos(x)$
- Để bắt đầu thuật toán, ta sẽ khởi tạo giá trị ban đầu của x bằng $\boldsymbol{x_0 = -5.}$ Sau đó, ta sẽ lặp lại việc cập nhật giá trị của x cho đến khi đạt được giá trị tối ưu. Cụ thể, ta sẽ lặp lại các bước sau cho đến khi đạo hàm của hàm số f(x) gần bằng 0:
    - Tính giá trị của đạo hàm f'(x) tại điểm x hiện tại.
    - Cập nhật giá trị của x bằng công thức $\boldsymbol{x_{new} = x - n * f'(x)}$
    - Quay lại bước 1.
- Minh hoạ khi ta thay đổi các giá trị cho hàm một biến khác nhưng thay đổi giá trị:



Hình 3 Gradient descent cho hàm một biến $f(x) = x^2 + 5\sin(x)\,; x_0 = 5; n = 0.10$



Hình 4 Gradient descent cho hàm một biến $f(x) = x^2 + 5\sin(x)\,; x_0 = -5; n = 0.01$

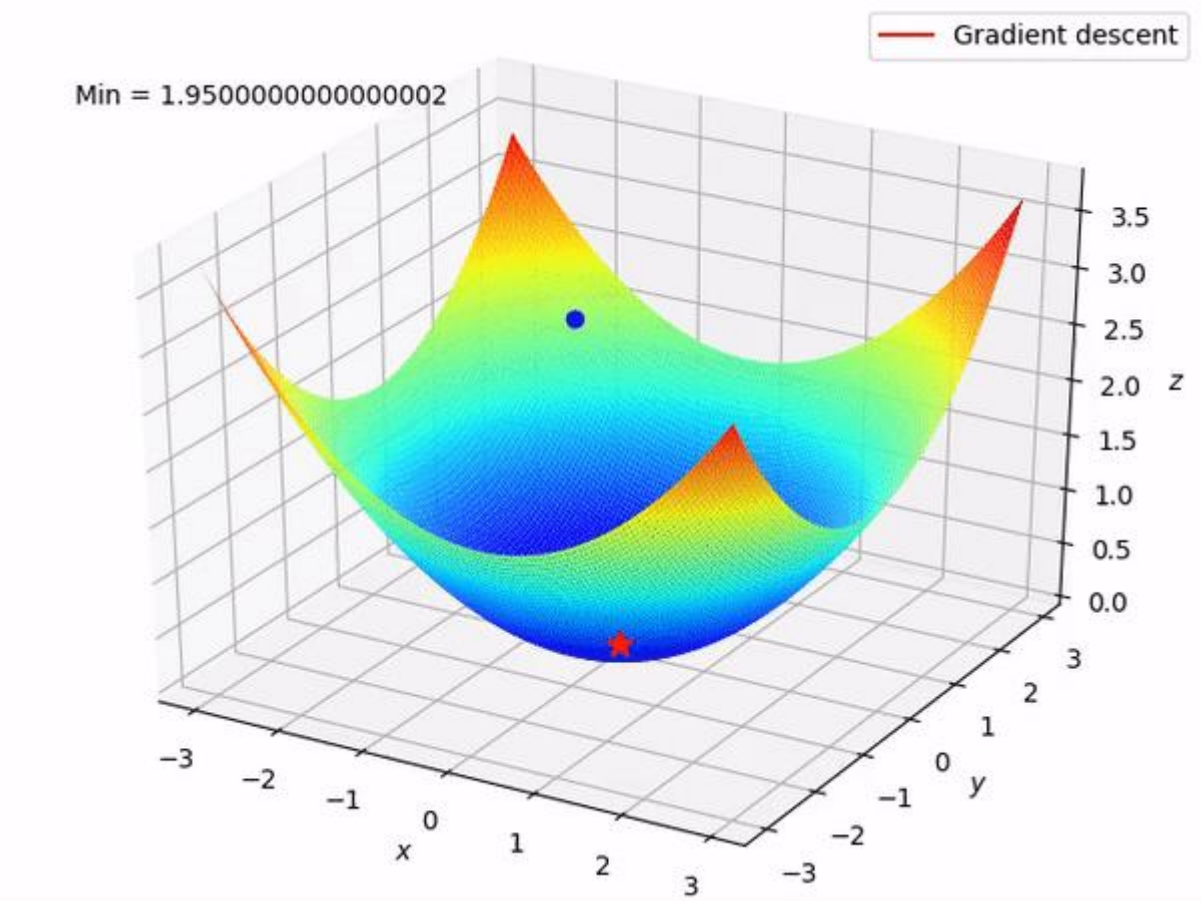


Hình 5 Gradient descent cho hàm một biến $f(x) = x^2 + 5\sin(x)\,; x_0 = -5; n = 0.5$

- Nguồn:
  - **Optimizer- Hiểu sâu về các thuật toán tối ưu (GD,SGD,Adam,..) (viblo.asia)**
  - https://machinelearningcoban.com/2017/01/12/gradientdescent/
- Qua các minh hoạ ta có thể nhận xét:
  - Điểm khởi tạo: lựa chọn điểm khởi tạo khác nhau sẽ ảnh hưởng đến kết quả cuối cùng, cũng như tác động trực tiếp đến quá trình hội tụ của chúng
  - Tốc độ học (learning rate): Tốc độ học quá nhỏ có thể làm chậm tốc độ hội tụ và kéo dài quá trình huấn luyện. Ngược lại, tốc độ học quá lớn có thể dẫn đến việc không hội tụ vì bước nhảy quá lớn, mô hình không thể tiến gần đến điểm cực tiểu.

1.2.1.2 Gradient descent cho hàm nhiều biến:

- Đối với hàm nhiều biến $f(\vec{x})$, thuật toán này sẽ tính grandient của hàm f, tại một điểm ngẫu nhiên $\vec{x}$, sau đó di chuyển $\vec{x}$ ngược hướng với gradient này.
- Công thức để 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 $\vec{x}$ bước thứ t
        - α *là learning rate (tốc độ học)*
        - $\nabla f(\vec{x}_t)$ *là gradient của hàm* f *tại* $\vec{x}_t$



Hình 6 Gradient descent cho hàm nhiều biến

1.2.1.3 Các thuật toán gradient descent:

- Có nhiều biến thể của thuật toán này tuỳ thuộc vào việc lựa chọn dữ liệu huấn luyện và thứ tự cập nhật:

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

1.2.1.4 Biến thể

- Một số cải tiến của Gradient Descent để tăng tốc độ hội tụ và giảm thiểu dao động trong quá trình hội tụ:
  - 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.

1.2.1.5 Đánh giá

- Ưu điểm:
  - Đơn giản: dễ hiểu và thực hiện.
  - Giải quyết vấn đề: Hiệu quả trong việc tối ưu hóa mô hình neural network bằng cách cập nhật trọng số sau mỗi vòng lặp.
- Nhược điểm:

- Phụ thuộc vào điểm khởi tạo và tốc độ học: Điểm khởi tạo ban đầu và tốc độ học có thể ảnh hưởng đến kết quả cuối cùng của thuật toán.
- Khó hội tụ đến điểm cực tiểu đúng: Đối với các hàm số có nhiều điểm cực tiểu, thuật toán có thể hội tụ đến điểm cực tiểu không mong muốn tùy thuộc vào điểm khởi tạo.
- Tốc độ học không cân đối: Tốc độ học quá lớn hoặc quá nhỏ đều có thể gây ra vấn đề, từ việc không hội tụ đến quá trình học chậm.

1.2.2 Stochastic Gradient Descent (SGD)

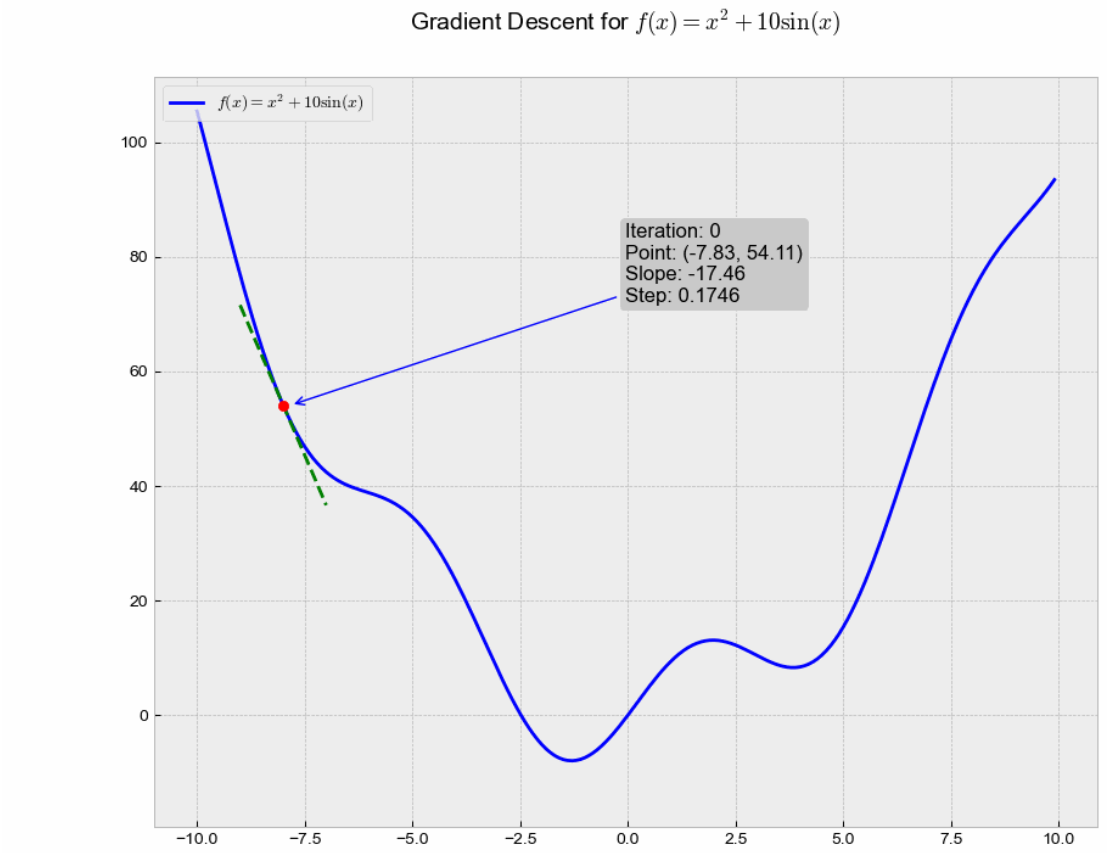
1.2.2.1 Stochastic Gradient Descent là gì?

- Stochastic Gradient Descent (SGD) là một biến thể của thuật toán Gradient Descent được sử dụng để tối ưu hóa các mô hình học máy. Nó giải quyết vấn đề hiệu suất tính toán của các phương pháp Gradient Descent truyền thống khi xử lý các bộ dữ liệu lớn trong các dự án học máy.

1.2.2.2 Stochastic Gradient Descent Algorithm

- Thuật toán được khởi tạo bằng cách thiết lập ngẫu nhiên các tham số của mô hình. Số lần lặp và tốc độ học (alpha) để cập nhật các tham số được xác định. Các bước sau được lặp lại cho đến khi mô hình hội tụ hoặc đạt tới số lần lặp tối đa:
  - Trộn tập dữ liệu huấn luyện để tạo sự ngẫu nhiên.
  - Lặp lại qua mỗi ví dụ huấn luyện (hoặc một batch nhỏ) theo thứ tự đã được trộn.
  - Tính toán đạo hàm của hàm mất mát đối với các tham số mô hình sử dụng ví dụ huấn luyện hiện tại (hoặc batch).
  - Cập nhật các tham số mô hình bằng cách di chuyển theo hướng đạo hàm âm, được tỷ lệ theo tốc độ học.
  - Đánh giá tiêu chí hội tụ, như sự khác biệt trong hàm mất mát giữa các lần lặp của đạo hàm.

- Trả về các tham số tối ưu hóa: Một khi tiêu chí hội tụ được đáp ứng hoặc số lần lặp tối đa đã đạt, trả về các tham số mô hình đã tối ưu hóa.
- Công thức để 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 $\vec{x}$ bước thứ t
        - α *là learning rate (tốc độ học)*
        - $\nabla f(\vec{x}_t)$ *là gradient của hàm* f *tại* $\vec{x}_t$

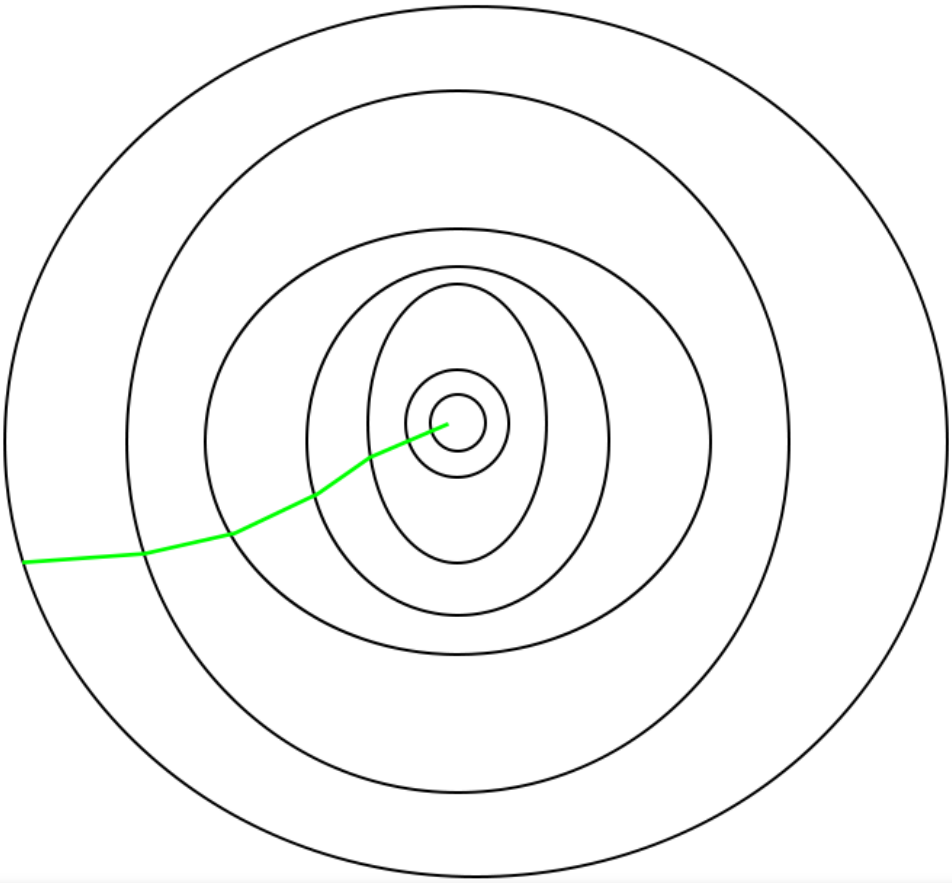


Hình 7 Minh họa thuật toán Stochastic Gradient Descent

- Trong SGD, do chỉ một mẫu từ tập dữ liệu được chọn ngẫu nhiên cho mỗi vòng lặp, đường đi của thuật toán để đạt đến điểm cực tiểu thường nhiễu hơn so với

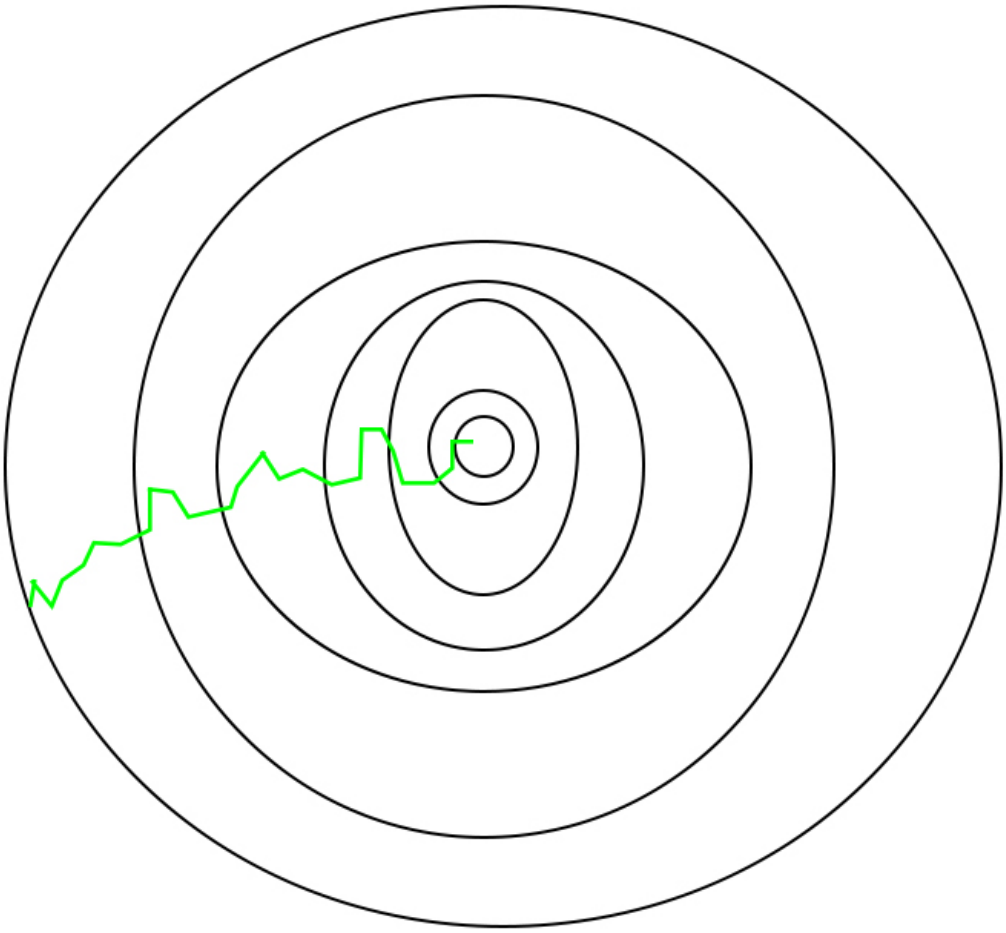
thuật toán Gradient Descent thông thường. Nhưng điều đó không quá quan trọng vì đường đi của thuật toán không ảnh hưởng nhiều, miễn là chúng ta đạt được điểm cực tiểu và với thời gian huấn luyện ngắn hơn đáng kể.

- So sánh bằng biểu đồ:
    - Giảm dần độ dốc ngẫu nhiên (SGD)



Hình 8 Batch gradient optimization path

    - Đường đi của Stochastic gradient Descent



Hình 9 stochastic gradient optimization path

- Tại sao lại sử dụng SGD thay vì GD mặc dù đường đi của nó khá rối? Vấn đề chính là GD gặp hạn chế khi xử lý cơ sở dữ liệu lớn (ví dụ: vài triệu dữ liệu) vì việc tính toán đạo hàm trên toàn bộ dữ liệu ở mỗi vòng lặp trở nên rất tốn kém.
- Giả sử bạn làm việc trong một công ty bán lẻ trực tuyến và muốn xây dựng một mô hình để dự đoán hành vi mua sắm của khách hàng dựa trên thông tin về lịch sử mua hàng, thời gian truy cập, và các yếu tố khác.
    - Gradient Descent (GD): Nếu bạn áp dụng GD để huấn luyện mô hình của mình, mỗi lần cập nhật tham số, bạn sẽ tính toán gradient của hàm mất mát trên toàn bộ lịch sử mua hàng của tất cả khách hàng. Điều này có thể

trở nên cồng kềnh khi bạn có hàng triệu thông tin mua hàng và muốn cập nhật mô hình theo thời gian thực khi có dữ liệu mới.

- Stochastic Gradient Descent (SGD): Ngược lại, nếu bạn sử dụng SGD, mỗi lần cập nhật tham số, bạn chỉ cần tính toán gradient trên một mẫu nhỏ hoặc một khách hàng cụ thể. Khi có dữ liệu mua hàng mới từ một khách hàng, bạn có thể nhanh chóng cập nhật mô hình chỉ bằng thông tin từ khách hàng đó mà không cần phải tính toán lại trên toàn bộ dữ liệu, giúp tiết kiệm thời gian tính toán và phù hợp với học trực tuyến.
- Ví dụ này minh họa rõ ràng sự khác biệt giữa GD và SGD khi áp dụng vào việc phân tích dữ liệu khách hàng và học trực tuyến để cập nhật mô hình một cách hiệu quả.

1.2.2.3 Đánh giá

- Ưu điểm:
  - Xử lý cơ sở dữ liệu lớn: SGD có khả năng làm việc với cơ sở dữ liệu lớn mà GD không thể.
  - Được sử dụng rộng rãi: Hiện nay, SGD vẫn là một trong những thuật toán tối ưu được sử dụng phổ biến.
- Nhược điểm:
  - Chưa giải quyết được các nhược điểm của Gradient Descent: Vẫn còn đối mặt với các hạn chế lớn của Gradient Descent như việc điều chỉnh learning rate và điểm dữ liệu ban đầu. Do đó, thường cần kết hợp SGD với các thuật toán khác như Momentum, AdaGrad, … để cải thiện hiệu suất. Các thuật toán này sẽ được đề cập trong phần sau.
- Việc kết hợp SGD với các thuật toán tối ưu khác thường giúp cải thiện khả năng hội tụ và hiệu suất của quá trình tối ưu hóa, giảm thiểu tác động của nhược điểm của SGD đơn lẻ.

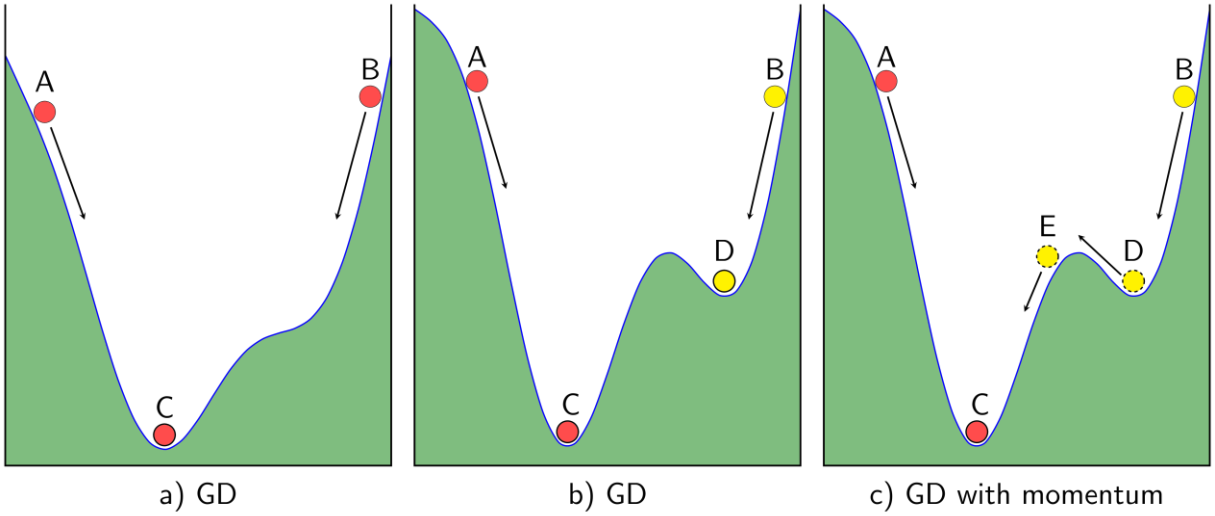
1.2.3 Momentum

1.2.3.1 Momentum là gì?

- Momentum là một khái niệm mô tả xu hướng của một đối tượng hoặc hệ thống tiếp tục di chuyển theo một hướng cụ thể do lực hoặc vận tốc tích lũy. Ví dụ, một đoàn tàu đang chuyển động có độ lớn của momentum lớn, khiến cho việc dừng đột ngột trở nên khó khăn. Trong học máy, momentum được sử dụng để tăng cường tốc độ hội tụ của các thuật toán tối ưu hóa bằng cách thêm một phần tử của cập nhật trước đó vào cập nhật hiện tại, cho phép thuật toán tăng tốc độ theo hướng hạ xuống nhanh nhất và làm giảm sự dao động.
- Trong học máy, momentum là một kỹ thuật được sử dụng để cải thiện tốc độ hội tụ của các thuật toán tối ưu hóa, như gradient descent, bằng cách thêm một phần tử của cập nhật trước đó vào cập nhật hiện tại. Điều này giúp thuật toán tăng tốc độ theo hướng hạ xuống nhiều nhất và làm giảm sự dao động, dẫn đến việc hội tụ nhanh hơn và cải thiện hiệu suất.
- Trong tối ưu hóa gradient descent, momentum hoạt động bằng cách thêm một phần tử của cập nhật trước đó vào cập nhật hiện tại. Điều này giúp thuật toán tăng tốc độ theo hướng hạ xuống nhanh nhất và làm giảm sự dao động. Bằng việc tích hợp momentum, thuật toán có thể hội tụ nhanh hơn và đạt được hiệu suất tốt hơn. Điều này đặc biệt hữu ích trong các mô hình học sâu, nơi việc huấn luyện có thể tốn thời gian và tài nguyên tính toán

1.2.3.2 Momentum dưới góc nhìn vật lý?

- Một cách khác để mô tả thuật toán Gradient Descent là so sánh với việc hòn bi lăn trên một địa hình giống như thung lũng, như hình 1a) dưới đây. Bất kể khởi điểm là A hay B, hòn bi cuối cùng sẽ lăn xuống và dừng lại tại vị trí C. Đây là một cách hình dung hữu ích để hiểu cách thuật toán hoạt động, khi nó luôn tìm đường đi xuống tới điểm thấp nhất trên địa hình của hàm mất mát.

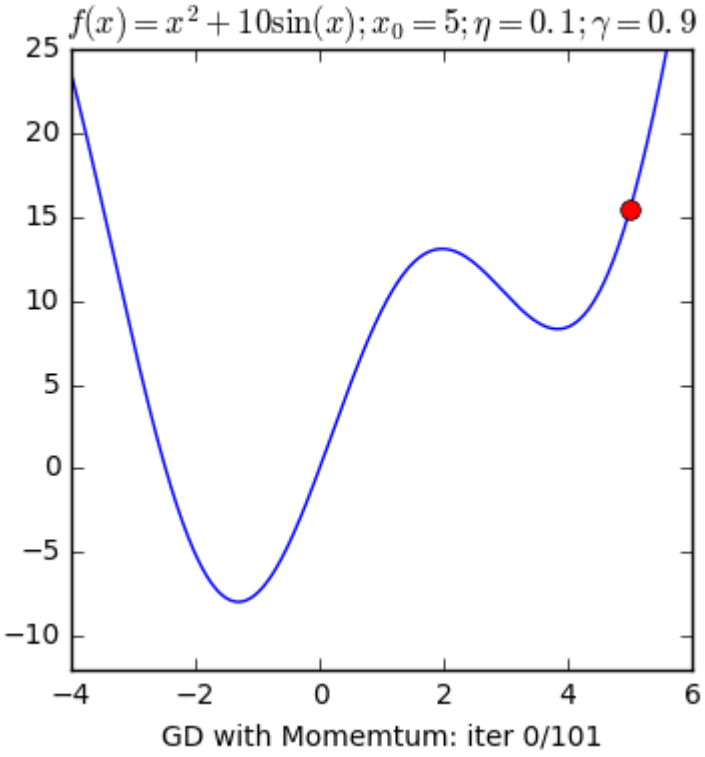


Hình 10 So sánh Gradient Descent với các hiện tượng vật lý

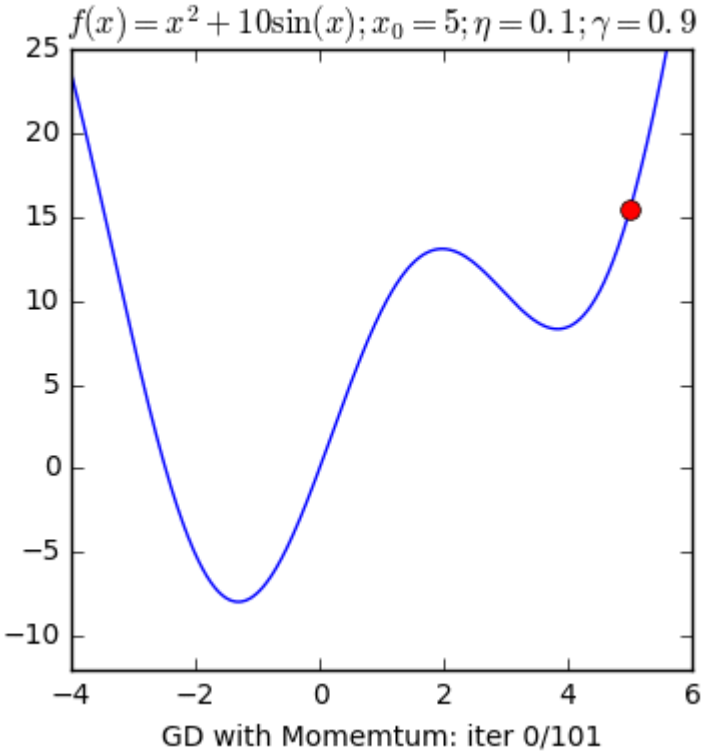
- Trong trường hợp bề mặt hàm mất mát có hai đỉnh thung lũng như Hình 1b), tùy thuộc vào điểm khởi đầu ở A hoặc B, hòn bi cuối cùng có thể dừng ở C hoặc D. Điểm D thường được gọi là điểm cực tiểu cục bộ, một vị trí mà thuật toán không muốn dừng lại.
- Nếu xem xét một cách vật lý hơn, trong Hình 1b), khi hòn bi từ B lăn đến D với vận tốc ban đầu đủ lớn, nó có thể tiếp tục đi lên dốc bên trái của D. Và với vận tốc ban đầu lớn hơn nữa, bi có thể vượt qua dốc lên E, sau đó lăn xuống C như trong Hình 1c). Điều này là điều mà chúng ta mong muốn. Tuy nhiên, việc bi từ A lăn tới C, sau đó tiếp tục lăn tới E rồi đến D khá khó xảy ra, bởi dốc CE cao hơn rất nhiều so với dốc DE.
- Dựa trên hiện tượng này, Momentum ra đời nhằm khắc phục việc Gradient Descent rơi vào điểm cực tiểu cục bộ không mong muốn. Momentum, như tên gọi của nó, giúp thuật toán "theo đà" để vượt qua những điểm cực tiểu cục bộ và tiếp tục hướng tới điểm cực tiểu toàn cục.

1.2.3.3 Momentum Algorithm

- Để biểu diễn momentum trong thuật toán Gradient Descent bằng toán học, chúng ta sử dụng một biến vận tốc để cập nhật vị trí mới của nghiệm. Nếu coi biến này như vận tốc trong vật lý, vị trí mới của nghiệm sẽ được xác định bằng công thức:
    - $\theta_{t+1} = \theta_t - v_t$
    - Trong đó dấu trừ thể hiện việc di chuyển ngược với đạo hàm.
- Để tính toán vận tốc v_t sao cho nó kết hợp thông tin của độ dốc ($\nabla_\theta J(\theta)$) và thông tin của đà (vận tốc trước đó v_{t-1}), chúng ta có thể sử dụng công thức:
    - $v_t = \gamma v_{t-1} + n \nabla_\theta J(\theta)$
    - Trong đó:
        - γ thường có giá trị khoảng 0.9
        - v_t là vận tốc tại thời điểm trước đó
        - $\nabla_\theta J(\theta)$ là độ dốc của điểm trước đó
        - n là learning rate.
- Sau đó, vị trí mới được cập nhật theo công thức: $\theta = \theta - v_t$
- Thuật toán này đơn giản nhưng rất hiệu quả trong thực tế, đặc biệt trong không gian đa chiều, với cách tính toán tương tự. Bên dưới là một ví dụ minh họa trong không gian một chiều.
- Ví dụ: Cùng nhìn vào một hàm đơn giản có đặc tính đáng chú ý:
    - $f(x) = x^2 + 10\sin(x)$ và đạo hàm $f'(x) = 2x + 10\cos(x)$
    - Hàm này có hai điểm local minimum, trong đó có một điểm là global minimum.
    - Hình dưới đây thể hiện sự khác nhau giữa thuật toán GD và thuật toán GD với Momentum:



Hình 11 Minh họa thuật toán GD



Hình 12 Minh họa thuật toán Momentum

1.2.3.4 Đánh giá:

- Ưu điểm:
  - Vượt qua cực tiểu cục bộ: Giúp thuật toán tối ưu hóa tiến tới điểm global minimum thay vì dừng lại ở các điểm cực tiểu cục bộ
- Nhược điểm:
  - Hậu quả của đà: Khi tiến gần tới điểm cuối cùng, việc tồn tại của đà có thể làm chậm quá trình dừng lại của thuật toán, tạo ra dao động qua lại và kéo dài thời gian dừng lại.

1.2.4 Adagrad

1.2.4.1 Adagrad là gì?

- Adagrad là một thuật toán tối ưu hóa phức tạp hơn có khả năng điều chỉnh tốc độ học tập cho từng tham số một cách riêng biệt. Điều này có nghĩa là cho mỗi

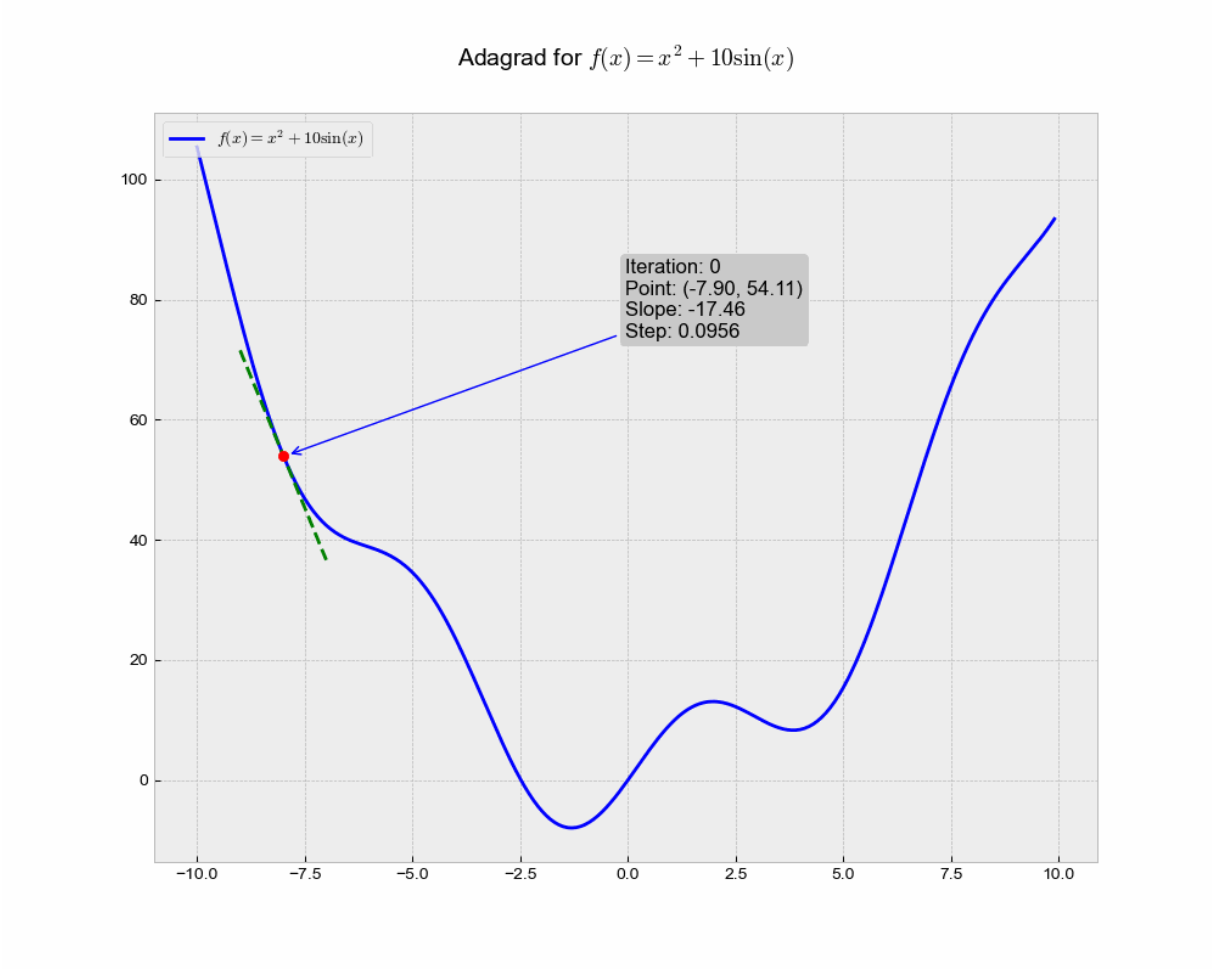
bước huấn luyện, mỗi tham số có thể có một tốc độ học tập khác nhau. Điều này có thể rất hữu ích khi xử lý dữ liệu thưa, vì Adagrad sẽ gán tốc độ học tập cao hơn cho các tham số hiếm xuất hiện.

- Khi nghiên cứu các biến thể của Gradient Descent, cần xem xét vai trò đặc biệt của Square Gradients trong các phương pháp như AdaGrad, nơi việc xử lý Các Đặc Trưng Hiếm Xuất hiện trở nên khả thi hơn nhờ khả năng thích nghi của tốc độ học tập theo giá trị lịch sử của Vector Độ Dốc.
- Adagrad điều chỉnh tốc độ học tập của tất cả các tham số mô hình, điều chỉnh chúng theo tỷ lệ nghịch với căn bậc hai của tổng bình phương độ dốc đã được tính trước đó (Duchi et al., 2011). Điều này có nghĩa là các tham số liên quan đến đạo hàm riêng lớn sẽ trải qua mức giảm tốc độ học tập nhanh hơn, trong khi những tham số có đạo hàm riêng nhỏ hơn sẽ trải qua mức giảm tốc độ học tập chậm hơn.

1.2.4.2 Adagrad Algorithm

- Thuật toán Adagrad điều chỉnh tốc độ học tập của mỗi tham số dựa trên lịch sử của độ dốc đã tính toán cho từng tham số. Nó áp dụng một tỷ lệ thích ứng cho mỗi tham số, cho phép việc học tập linh hoạt hơn cho các tham số có độ dốc khác nhau.
- Công thức cập nhật của Adagrad cho mỗi tham số $\theta\_\{i\}$ tại bước thời gian t trong quá trình huấn luyện được biểu diễn như sau:
    - $\theta_{t+1,i} = \theta_{t,i} - \frac{n}{\sqrt{r_{t,ii}+\epsilon}} \nabla j(\theta_{t,i})$
    - Trong đó:
        - $\theta_{t+1,i}$ là giá trị cập nhật mới của tham số θ_i tại thời gian $t+1$
        - $\theta_{t,i}$ là giá trị hiện tại của tham số θ_i tại thời gian t
        - n là learning rate.

- $\sqrt{r_{t,ii}+\in}$ là căn bậc hai của tổng tích lũy của bình phương độ dốc cho tham số θ_i đến bước thời gian t cộng với một giá trị dương nhỏ ϵ để tránh chia cho số không.
- $\nabla j(\theta_{t,i})$ là độ dốc của hàm chi phí J theo tham số θ_i tại bước thời gian t
- Gía trị $r_{t,ii}$ được tính theo công thức sau: $r_{t,ii} = r_{t-1,ii} + \left(\nabla j(\theta_{t,i})\right)^2$



Hình 13 Minh hoạ cho thuật toán Adagrad

1.2.4.3 Đánh giá:

- Ưu điểm:
    - Adagrad tự động điều chỉnh tốc độ học tập cho từng tham số một cách tự động, loại bỏ nhu cầu phải thay đổi tốc độ học tập bằng tay.
- Nhược điểm:
    - Tính chất tích lũy của bình phương gradient dần tăng theo thời gian, có thể làm giảm tốc độ học tập và làm chậm quá trình huấn luyện.

1.2.5 RMSprop

1.2.5.1 RMSprop là gì?

- RMSprop, giống như các thuật toán gradient descent khác, hoạt động bằng cách tính đạo hàm của hàm mất mát đối với các tham số của mô hình và điều chỉnh các tham số này theo hướng ngược lại của đạo hàm để giảm thiểu mất mát. Tuy nhiên, RMSProp tích hợp một số kỹ thuật bổ sung để cải thiện hiệu suất của quá trình tối ưu hóa.
- Một điểm quan trọng là việc sử dụng trung bình di động của bình phương các đạo hàm để điều chỉnh tốc độ học cho mỗi tham số. Điều này giúp ổn định quá trình học tập, ngăn chặn sự dao động trong quỹ đạo tối ưu hóa.

1.2.5.2 RMSprop Algorithm

- Thuật toán RMSprop là một thuật toán tối ưu hóa gradient được sử dụng trong quá trình huấn luyện mô hình machine learning. Nó điều chỉnh tốc độ học (learning rate) cho từng tham số của mô hình dựa trên trung bình di động của bình phương gradient.
- Công thức của RMSprop thực hiện việc điều chỉnh tốc độ học η cho từng tham θ_i số theo cách sau:
    - Cập nhật trung bình di động của bình phương gradient:
        - $E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta)\left(\nabla J(\theta_t)\right)^2$
    - Cập nhật tham số theo learning rate đã điều chỉnh:
        - $\theta_{t+1} = \theta_t - \frac{n}{\sqrt{E[g^2]_t + \epsilon}} \nabla \mathrm{j}(\theta_t)$

- Trong đó:
  - β là hệ số giảm thường đượt đặt ở giá trị khoảng 0.9
  - $E[g^2]_t$ là trung bình di động của bình phương gradient tại t
  - $\nabla J(\theta_t)$ là grandient hàm mất mát J theo tham số θ_t tại thời điểm t
  - n là learning rate
  - $\in$ là một giá trị (thường 10^{-8})

1.2.5.3 Đánh giá:

- Ưu điểm:
  - RMSprop là khắc phục được vấn đề tốc độ học giảm dần của Adagrad. Trong Adagrad, tốc độ học giảm theo thời gian có thể làm chậm quá trình huấn luyện và khiến mô hình dễ bị đóng băng. RMSprop giúp ổn định quá trình huấn luyện này hơn bằng cách điều chỉnh learning rate một cách thông minh.
- Nhược điểm:
  - RMSprop là có thể dẫn đến nghiệm chỉ ở local minimum thay vì global minimum như thuật toán Momentum. Điều này có thể khiến cho mô hình không đạt được sự tối ưu tốt nhất. Vì vậy, người ta thường kết hợp cả hai thuật toán Momentum và RMSprop để tạo ra thuật toán tối ưu hóa Adam, một phương pháp tối ưu hóa kết hợp từ cả hai. Sự kết hợp này cung cấp sự cân bằng giữa tốc độ học và khả năng tìm nghiệm tốt hơn.

1.2.6 Adam

1.2.6.1 Adam là gì?

- Adam, viết tắt của Adaptive Moment Estimation, là một thuật toán tối ưu hóa được sử dụng trong lĩnh vực học máy để cập nhật trọng số mạng một cách lặp lại dựa trên dữ liệu huấn luyện. Adam được coi là một phương pháp mở rộng của gradient descent ngẫu nhiên và nổi tiếng với khả năng xử lý hiệu quả độ dốc thưa cũng như tính ổn định với việc lựa chọn siêu tham số.

1.2.6.2 Adam Algorithm

- Thuật toán Adam (Adaptive Moment Estimation) là một phương pháp tối ưu hóa được sử dụng rộng rãi trong huấn luyện mô hình học máy. Nó kết hợp giữa hai khía cạnh quan trọng trong việc cập nhật trọng số của mô hình: hướng dẫn tốc độ học theo mỗi tham số và điều chỉnh tốc độ học dựa trên lịch sử của gradient.
- Công thức cập nhật trong thuật toán Adam là:
  - $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
  - $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  - $\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$
  - $\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$
  - $\theta_{t+1} = \theta_t - \frac{n}{\sqrt{\widehat{v_t}} + \in} \widehat{m_t}$
  - Trong đó:
    - m_t và v_t là các giá trị ước lượng của first moment (mean) và second moment (uncentered variance) của gradient tại bước thời gian t
    - g_t là gradient tại thời điểm t
    - $\widehat{m_t}$ và $\widehat{v_t}$ là các phiên bản được điều chỉnh của first moment và second moment để khắc phục sự khác biệt ban đầu của chúng.
    - θ_t và θ_{t+1}là các tham số của mô hình tại thời điểm t và $t+1$
    - n là learning rate, β_1 và β_2 là các hệ số decay rates.
    - $\in$ là một giá trị (thường 10^{-8})
  - Phân tích công thức:
    - Adam tính toán hai giá trị ước lượng m_t và v_t để theo dõi trung bình cục bộ của gradient và độ biến đổi của gradient.
    - Các giá trị được điều chỉnh $\widehat{m_t}$ và $\widehat{v_t}$ giúp khắc phục việc khởi tạo ban đầu của là m_t và v_t là 0.

- Công thức cuối cùng điều chỉnh tham số của mô hình dựa trên các ước lượng này, điều chỉnh learning rate dựa trên v_t và thực hiện cập nhật với hướng của m_t. Điều này giúp ổn định tốc độ học và cập nhật hiệu quả trọng số của mô hình.

1.2.6.3 Đánh giá:

- Ưu điểm:
  - Hiệu suất đa dạng: Adam thường hoạt động hiệu quả với nhiều loại dữ liệu khác nhau và không đòi hỏi nhiều tuỳ chỉnh hyperparameters như các thuật toán khác.
  - Xử lý gradient thưa: Nó có khả năng xử lý hiệu quả các gradient thưa (sparse gradients), phù hợp với các mô hình sử dụng sparse data như trong mạng nơ-ron ngôn ngữ.
  - Tính ổn định: Thuật toán này có khả năng giữ cho tốc độ học ổn định và hiệu quả ở các bước cập nhật khác nhau, giúp mô hình học nhanh và ổn định hơn.
- Nhược điểm:
  - Vấn đề tốc độ học giảm dần: Tương tự như Adagrad, Adam có thể gặp vấn đề với việc tốc độ học giảm dần theo thời gian. Điều này có thể dẫn đến việc training bị chậm lại đến mức đóng băng.
  - Chỉ hội tụ đến local minimum: Adam có thể chỉ dừng ở local minimum thay vì global minimum. Kết hợp với thuật toán khác như Momentum giúp cải thiện khả năng hội tụ đến global minimum.

1.3 So sánh tổng thể các thuật toán optimizer:

Thuật toán	Ưu điểm	Nhược điểm
Gradient Descent	Dễ hiểu và dễ cài đặt	Chậm và dễ bị mắc kẹt ở điểm cực tiểu địa phương

Stochastic Gradient Descent	Nhanh và hiệu quả	Không ổn định và có thể bị mắc kẹt ở điểm cực tiểu địa phương
Momentum	Giúp tăng tốc độ hội tụ	Có thể bị mắc kẹt ở điểm cực tiểu địa phương
Adagrad	Tự động điều chỉnh tỷ lệ học tập	Có thể bị mắc kẹt ở điểm cực tiểu địa phương
RMSprop	Hiệu quả và ổn định	Có thể bị mắc kẹt ở điểm cực tiểu địa phương
Adam	Hiệu quả và ổn định	Có thể bị mắc kẹt ở điểm cực tiểu địa phương

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:

2.1Continual Learning

"Continual Learning" là một khái niệm trong lĩnh vực học máy, nó liên quan đến việc cập nhật và mở rộng kiến thức của một mô hình máy học khi có thêm dữ liệu mới được cung cấp. Điều này giúp cho mô hình có thể cải thiện hiệu suất của nó theo thời gian và đáp ứng tốt hơn với các tình huống mới. Nó khác với "Continuous Learning" - một khái niệm khác trong lĩnh vực học máy - bởi vì "Continuous Learning" đề cập đến việc mô hình máy học có thể học hỏi và cải thiện hiệu suất của nó mà không cần phải có dữ liệu mới.

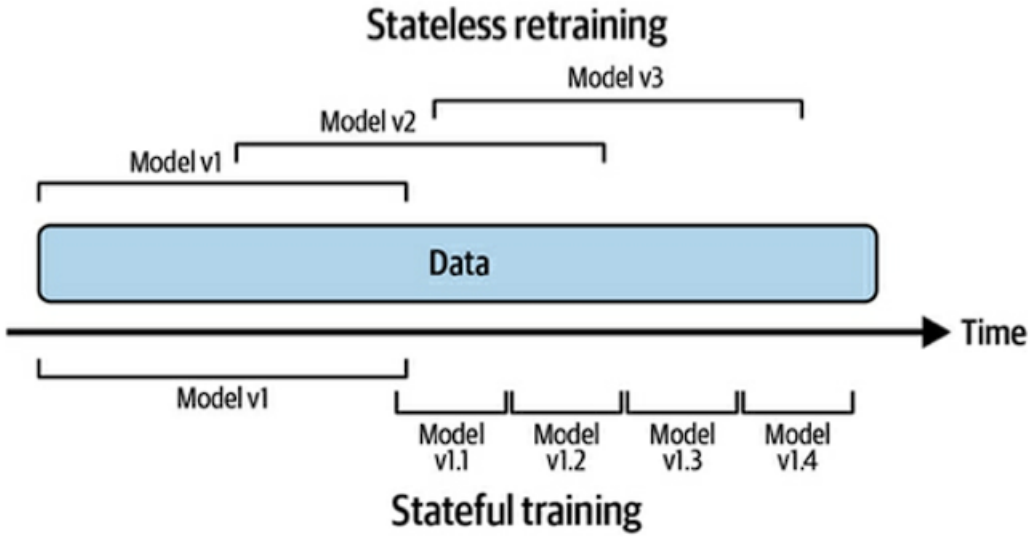
Một ví dụ về Continual Learning trong lĩnh vực học máy là khi bạn xây dựng một mô hình máy học để phân loại các loại hoa dựa trên các đặc trưng của chúng. Bạn đã huấn luyện mô hình của mình trên một tập dữ liệu ban đầu và đã đạt được độ chính xác tốt. Tuy nhiên, sau đó, bạn nhận được thêm dữ liệu mới về các loại hoa khác, và bạn muốn cập nhật mô hình của mình để có thể phân loại chúng một cách chính xác. Để làm điều này, bạn có thể sử dụng Continual Learning để cập nhật mô hình của mình với dữ liệu mới mà không cần phải huấn luyện lại từ đầu. Bằng cách này, mô hình của bạn có thể cải thiện hiệu suất của nó theo thời gian và đáp ứng tốt hơn với các tình huống mới.

2.1.1 Tại sao phải Continual Learning

- Lý do cơ bản cho điều này là giúp mô hình của bạn đồng bộ với các biến đổi trong phân phối dữ liệu. Có một số trường hợp sử dụng mà việc thích nghi nhanh chóng với sự thay đổi về phân phối là cực kỳ quan trọng. Dưới đây là một số ví dụ:

- Các trường hợp có thể xảy ra sự thay đổi không lường trước và nhanh chóng: Các trường hợp như chia sẻ dịch vụ đi lại thường chịu ảnh hưởng này. Ví dụ, có thể có một buổi hòa nhạc diễn ra ở một khu vực bất kỳ vào một ngày thứ Hai ngẫu nhiên và "mô hình giá thứ Hai" có thể không thích hợp để xử lý điều này.
- Các trường hợp mà không thể thu thập dữ liệu huấn luyện cho một sự kiện cụ thể: Một ví dụ về điều này là các mô hình thương mại điện tử vào ngày Black Friday hoặc một sự kiện giảm giá khác chưa từng được thử. Rất khó để thu thập dữ liệu lịch sử để dự đoán hành vi người dùng vào ngày Black Friday, do đó mô hình của bạn phải thích nghi trong suốt ngày.
- Các trường hợp nhạy cảm 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 thích nghi mô hình ngay khi bạn nhận được một số dữ liệu từ người dùng đó, bạn sẽ không thể đề xuất những điều liên quan cho người dùng đó.

2.1.2 Các khái niệm: Stateless retraining và Stateful training



Hình 14 Stateless retraining VS Stateful training

2.1.2.1 Stateless retraining

- Đào tạo lại mô hình của bạn 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 liên tục bằng cách sử dụng việc huấn luyện không trạng thái.

2.1.2.2 Stateful training (aka fine-tuning, incremental learning)

- Khởi tạo mô hình của bạn với trọng số từ vòng huấn luyện trước đó và tiếp tục huấn luyện bằng dữ liệu mới chưa được sử dụng.
    - 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 tài nguyên tính toán hơn.
        - Một số công ty đã báo cáo giảm 45% tài nguyên tính toán.
    - Điều này có thể loại bỏ hoàn toàn việc lưu trữ dữ liệu sau khi dữ liệu đã được sử dụng cho quá trình huấn luyện (và để lại một khoảng thời gian an

toàn). Theo lý thuyết điều này loại bỏ mối quan tâm về quyền riêng tư dữ liệu.

- Trong thực tế, hầu hết các công ty thường giữ lại mọi dữ liệu mặc cho việc không còn cần thiết.

- Đôi khi bạn sẽ cần chạy lại huấn luyện không trạng thái với một lượng lớn dữ liệu để hiệu chỉnh lại mô hình.
- Một khi hệ thống được thiết lập đúng, việc chuyển từ huấn luyện không trạng thái sang huấn luyện có trạng thái trở nên dễ dàng chỉ với một nút nhấn.
- Quá trình huấn luyện có trạng thái thường được sử dụng để tích hợp dữ liệu mới vào kiến trúc mô hình hiện tại và cố định (tức là lặp lại dữ liệu). Nếu bạn muốn thay đổi các đặc trưng hoặc kiến trúc của mô hình, bạn sẽ cần thực hiện quá trình huấn luyện không trạng thái lần đầu.

2.1.3 Các khái niệm: Tái sử dụng feature thông qua log and wait.

- Các đặc trưng được tính toán để sử dụng trong việc suy luận. Một số công ty lưu trữ các đặc trưng đã tính toán cho mỗi mẫu dữ liệu để có thể tái sử dụng chúng trong quá trình huấn luyện liên tục và qua đó tiết kiệm một phần tính toán. Phương pháp này được gọi là "log and wait."

2.1.4 Thách thức của Continual Learning

- Continual Learning đã được áp dụng trong công nghiệp với thành công lớn. Tuy nhiên, nó có 3 thách thức lớn mà các tổ chức cần phải vượt qua:

2.1.4.1 Fresh data access challenge

- Nếu bạn muốn cập nhật mô hình mỗi giờ, bạn cần dữ liệu huấn luyện có nhãn chất lượng mỗi giờ. Càng ngắn khoảng thời gian cập nhật, thách thức này càng trở nên quan trọng hơn.
- Vấn đề: Tốc độ lưu trữ dữ liệu vào kho dữ liệu:

- Nhiều tổ chức lấy dữ liệu training từ kho dữ liệu của họ như Snowflake hoặc BigQuery. Tuy nhiên, dữ liệu đến từ các nguồn khác nhau sẽ được gửi vào kho bằng các cơ chế khác nhau và ở tốc độ khác nhau.
- Một cách tiếp cận phổ biến để giải quyết vấn đề này là lấy dữ liệu trực tiếp từ quá trình vận chuyển theo thời gian thực để training trước khi đưa vào kho.

- Vấn đề: Tốc độ gán nhãn
  - Tốc độ gán nhãn dữ liệu mới thường là điểm bottleneck. Các nhiệm vụ có nhãn tự nhiên và chu kỳ phản hồi ngắn là những ứng viên tốt nhất cho continual learning (càng ngắn chu kỳ phản hồi thì có thể gán nhãn càng nhanh).
  - Nếu nhãn tự nhiên không dễ dàng thu được trong khoảng thời gian cần thiết, có thể thử các kỹ thuật giám sát yếu hoặc bán giám sát để thu được chúng đúng thời hạn (với chi phí có thể là nhãn nhiễu). Như một phương án cuối cùng, có thể xem xét gán nhãn chú thích đa phần và nhanh chóng từ cộng đồng.
  - Một yếu tố khác ảnh hưởng đến tốc độ gán nhãn là chiến lược tính nhãn: tính nhãn theo số lượng lớn. Những công việc này thường chạy định kỳ trên dữ liệu đã được lưu trữ trong kho dữ liệu. Tốc độ gán nhãn là một hàm số của tốc độ lưu trữ dữ liệu và tần suất của công việc tính nhãn. Tương tự như giải pháp trên, một phương pháp phổ biến để tăng tốc độ gán nhãn là tính nhãn từ giao thông vận tải thời gian thực (sự kiện) trực tiếp. Phép tính dòng này có những thách thức riêng của nó.

2.1.4.2 Evaluation Challenge

- Việc áp dụng continual learning như một thực hành xảy ra với rủi ro thất bại lớn. Càng thường xuyên cập nhật mô hình, cơ hội để mô hình của bạn thất bại càng nhiều.

- Ngoài ra, continual learning mở ra cánh cửa cho các cuộc phản đối được phối hợp để làm nhiễu mô hình. Điều này có nghĩa là kiểm thử mô hình của bạn trước khi triển khai rộng rãi là rất quan trọng.
- Kiểm thử mất thời gian, vì vậy đây có thể là một yếu tố giới hạn khác về tần suất cập nhật mô hình nhanh nhất mà bạn có thể đạt được. Ví dụ: Một mô hình mới cho phát hiện gian lận có thể mất khoảng 2 tuần để có đủ lưu lượng truy cập để đánh giá một cách tự tin.

2.1.4.3 Data scaling challenge

- Phép tính đặc trưng thường đòi hỏi phải có Scaling. Scaling yêu cầu truy cập vào thống kê dữ liệu toàn cầu như giá trị tối thiểu, tối đa, trung bình và phương sai.
- Nếu bạn đang sử dụng việc training không trạng thái, thống kê toàn cầu phải xem xét cả dữ liệu trước đó đã được sử dụng để training mô hình cùng với dữ liệu mới đang được sử dụng để làm mới nó. Việc theo dõi thống kê toàn cầu trong tình huống này có thể là khó khăn.
- Một kỹ thuật thông thường để thực hiện điều này là tính toán hoặc xấp xỉ các thống kê này một cách tăng dần khi bạn quan sát dữ liệu mới (không phải tải toàn bộ tập dữ liệu vào thời gian huấn luyện và tính toán từ đó).
- Một ví dụ về kỹ thuật này là "Xấp xỉ phân vị tối ưu trong Luồng dữ liệu". StandardScaler trong Sklearn có phương thức partial_fit cho phép một trình tự hóa đặc trưng được sử dụng với thống kê liên tục. Tuy nhiên, các phương thức tích hợp này chậm và không hỗ trợ một loạt các thống kê liên tục.

2.1.4.4 Algorithm challenge

- Thách thức này xuất hiện khi bạn sử dụng một số loại thuật toán và muốn cập nhật chúng rất nhanh (ví dụ: mỗi giờ).
- Các thuật toán ở đây là những thuật toán, theo thiết kế, phụ thuộc vào việc truy cập vào toàn bộ tập dữ liệu để được huấn luyện. Ví dụ, các mô hình dựa trên ma

trận, giảm chiều dữ liệu và cây quyết định. Các loại mô hình này không thể được training tăng dần với dữ liệu mới như mạng nơ-ron hoặc các mô hình dựa trên trọng số khác có thể.

- Ví dụ: Bạn không thể làm giảm chiều dữ liệu PCA tăng dần. Bạn cần toàn bộ tập dữ liệu.
- Thách thức chỉ xảy ra khi bạn cần cập nhật chúng rất nhanh vì bạn không thể chờ đợi cho thuật toán xử lý toàn bộ tập dữ liệu.
- Có một số biến thể của các mô hình bị ảnh hưởng đã được thiết kế để được huấn luyện tăng dần, nhưng việc áp dụng các thuật toán này không phổ biến rộng rãi. Một ví dụ là Hoeffding Trees và các biến thể con.

2.1.5 Các bước của Continual Learning

Các tổ chức thường hướng tới việc học tập liên tục theo bốn giai đoạn:

- ***Giai đoạn 1: Training thủ công***

  Các mô hình chỉ được đào tạo lại khi đáp ứng hai điều kiện:

  - (1) Hiệu suất của mô hình đã suy giảm đến mức hiện tại nó gây hại nhiều hơn là có lợi.
  - (2) Có thời gian để cập nhật mô hình.

- ***Giai đoạn 2: Training không trạng thái tự động theo lịch trình cố định***

  - Giai đoạn này thường xảy ra khi các mô hình chính của một miền đã được phát triển và do đó ưu tiên của bạn không còn là tạo các mô hình mới mà là duy trì và cải thiện các mô hình hiện có.
  - Tần suất đào tạo lại ở giai đoạn này thường dựa trên "gut feeling".
  - Điểm uốn giữa giai đoạn 1 và giai đoạn 2 thường là một tập lệnh để chạy quá trình đào tạo lại không trạng thái theo định kỳ. Việc viết tập lệnh này có thể rất dễ hoặc rất khó tùy thuộc vào số lượng phần phụ thuộc cần được phối hợp để đào tạo lại một mô hình.
  - Các bước cấp cao của tập lệnh này là:

        - Kéo dữ liệu.
        - Giảm mẫu hoặc lấy mẫu dữ liệu nếu cần thiết.
        - Trích xuất các tính năng.
        - Xử lý và/hoặc chú thích nhãn để tạo dữ liệu đào tạo.
        - Bắt đầu quá trình đào tạo.
        - Đánh giá mô hình mới.
        - Triển khai nó.
    - Có 2 phần cơ sở hạ tầng bổ sung mà bạn sẽ cần để triển khai tập lệnh:
        - Bộ lập lịch
        - Một cửa hàng mô hình để tự động tạo phiên bản và lưu trữ tất cả các thành phần cần thiết để tái tạo mô hình. Các cửa hàng mô hình trưởng thành là AWS SageMaker và MLFlow của Databrick.
- ***Giai đoạn 3: Training trạng thái tự động theo lịch trình cố định***
    - Để đạt được điều này, bạn cần phải cấu hình lại tập lệnh của mình và cách theo dõi dòng dữ liệu cũng như mô hình của bạn. Một ví dụ về phiên bản dòng dõi mô hình đơn giản:
        - V1 và 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 đào tạo lại không trạng thái hoàn toàn và V2 đang ở lần lặp thứ 3.
        - V1.2.12 so với V2.3.43 có nghĩa là đã có 12 khóa đào tạo trạng thái được thực hiện trên V1.2 và 43 khóa đào tạo được thực hiện trên V2.3.
        - Bạn có thể sẽ cần sử dụng điều này cùng với các kỹ thuật lập phiên bản khác như lập phiên bản dữ liệu để theo dõi bức tranh đầy đủ về cách các mô hình đang phát triển.

- Tại bất kỳ thời điểm nào, bạn sẽ có nhiều mô hình chạy trong sản xuất cùng lúc thông qua các sắp xếp được mô tả trong Testing models in Production.

- ***Giai đoạn 4: Continual Learning***

  Trong giai đoạn này, phần lịch trình cố định của các giai đoạn trước được thay thế bằng một số cơ chế kích hoạt đào tạo lại. Các tác nhân kích hoạt có thể là:

  - Dựa trên thời gian
  - Dựa trên hiệu suất
  - Dựa trên khối lượng
  - Dựa trên sự trôi dạt

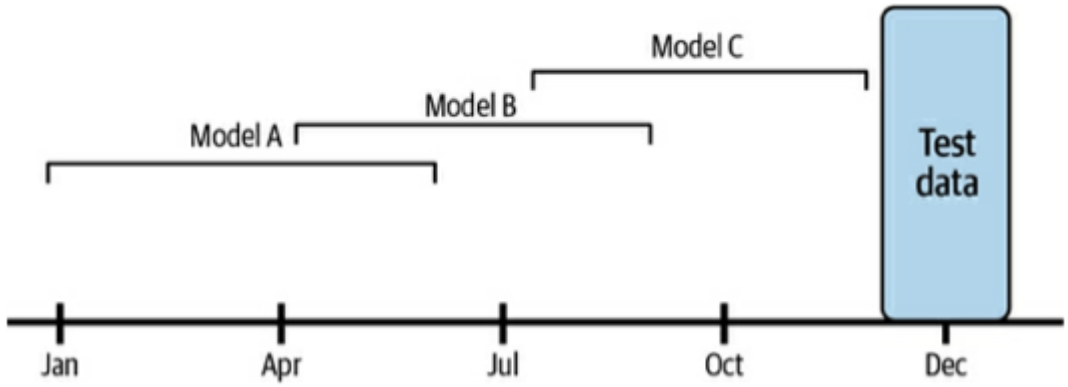
2.1.6 Cải thiện models

Trước tiên xác định mức lợi ích bạn nhận được khi cập nhật mô hình của mình với dữ liệu mới. Càng đạt được nhiều thì càng phải đào tạo lại thường xuyên.

2.1.6.1 Đo lường giá trị của độ mới dữ liệu

Một cách để định lượng giá trị của dữ liệu mới hơn là training 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 gán nhãn hiện tại.

Nếu 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 đào tạo lại sau chưa đầy 3 tháng.



Hình 15 Hiển thị các ví dụ về tập dữ liệu nhiều tháng

2.1.6.2 Khi nào nên thực hiện lặp lại mô hình?

Hầu hết phần này đều đề cập đến việc cập nhật mô hình của bạn với dữ liệu mới (tức là lặp lại dữ liệu). Tuy nhiên, trong thực tế, đôi khi bạn cần lặp lại mô hình. Dưới đây là một số gợi ý về thời điểm nên và không nên xem xét thực hiện lặp lại mô hình:

Nếu bạn tiếp tục giảm kích hoạt đào tạo lại việc lặp lại dữ liệu và không thu được nhiều lợi ích, nên tìm kiếm một mô hình tốt hơn.

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.

Câu hỏi "khi nào thực hiện lặp lại mô hình và lặp lại dữ liệu" vẫn chưa được nghiên cứu trả lời rõ ràng cho tất cả các nhiệm vụ. Cần phải chạy thử nghiệm nhiệm vụ cụ thể của mình để tìm ra thời điểm thực hiện việc đó.

2.2 Testing models in Production

Để kiểm tra đầy đủ các mô hình của bạn trước khi phổ biến rộng rãi, bạn cần cả **đánh giá ngoại tuyến trước khi triển khai** và **thử nghiệm trong sản xuất**.

2.2.1 Đánh giá ngoại tuyến trước khi triển khai

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

- Phần tách thử nghiệm 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.
- Kiểm tra ngược là ý tưởng sử dụng dữ liệu được gắn nhãn mới nhất mà mô hình chưa thấy trong quá trình đào tạo để kiểm tra hiệu suất (ví dụ: nếu bạn 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 Testing in Production Strategies

2.2.2.1 Shadow Deployment

- **Triển khai:** Triển khai mô hình người thách đấu song song với mô hình nhà vô địch hiện có. Gửi mọi yêu cầu đến cả hai mô hình nhưng chỉ phục vụ suy luận của mô hình vô địch. 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 của bạn có lỗi, dự đoán sẽ không được đưa ra.
  - Đơn giản về mặt khái niệm.
  - Thử nghiệm của bạn 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 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 khi chạy vì nó tăng gấp đôi số lượng dự đoán và do đó số lượng tính toán cần thiết.

2.2.2.2 A/B Testing

- **Triển khai:** triển khai mô hình người thách thức cùng với champion model (mô hình A) và định tuyến phần trăm lưu lượng truy cập đến challenger (mô hình B). Dự đoán từ challenger 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 hiệu suất của challenger có tốt hơn về mặt thống kê so với nhà vô địch hay không.
    - Một số trường hợp sử dụng không phù hợp với ý tưởng phân chia lưu lượng truy cập và có nhiều mô hình cùng một lúc. Trong những trường hợp này, thử nghiệm A/B có thể được thực hiện bằng cách thực hiện phân chia theo thời gian: mô hình A ngày hôm nay, mô hình B ngày hôm sau.
    - Việc phân chia lưu lượng truy cập phải là một thử nghiệm thực sự ngẫu nhiên. Nếu bạn đưa ra bất kỳ sự thiên vị lựa chọn nào về việc ai sẽ nhận được mô hình A so với mô hình B, kết luận của bạn sẽ không chính xác.
    - Thí nghiệm phải chạy đủ lâu để thu thập đủ mẫu nhằm đạt được độ tin cậy thống kê đủ lớn về sự khác biệt.
    - Ý nghĩa thống kê không phải là fool proof (đó là lý do tại sao nó có độ tin cậy). Nếu không có sự khác biệt thống kê giữa A và B, bạn có thể sử dụng cả hai.
    - Không có gì ngăn cản bạn chạy thử nghiệm A/B/C/D nếu bạn muốn.
- **Ưu điểm:**

Vì dự đoán được cung cấp cho người dùng nên kỹ thuật này cho phép bạn nắm bắt đầy đủ cách người dùng phản ứng với các mô hình khác nhau.

Thử nghiệm A/B rất dễ hiểu và có rất nhiều thư viện cũng như tài liệu xung quanh nó.

Việc chạy này rẻ vì chỉ có một dự đoán cho mỗi yêu cầu.

Bạn sẽ 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).

- **Nhược điểm:**
    - Nó kém an toàn hơn so với shadow deployment.
    - Bạn có quyền lựa chọn 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.

2.2.2.3 Canary Release

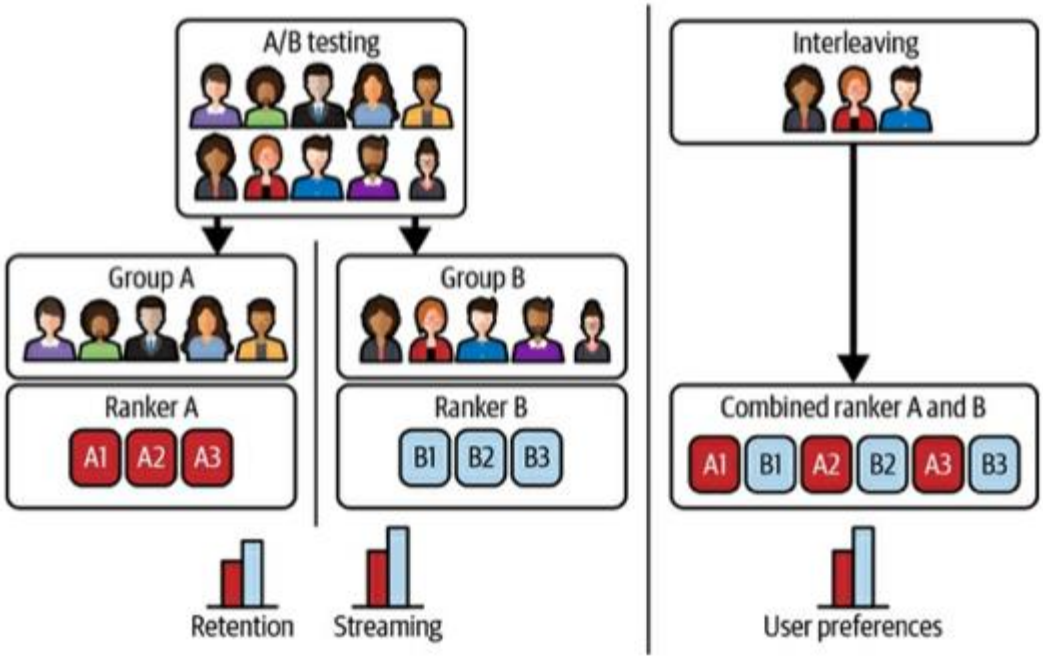
- **Triển khai:** triển khai challenger và champion cạnh nhau nhưng bắt đầu với challenger không tham gia traffic. Từ từ di chuyển lưu lượng truy cập từ nhà vô địch đến challenger. Theo dõi các số liệu hiệu suất của challenger, nếu chúng trông ổn, hãy tiếp tục cho đến khi tất cả lưu lượng truy cập đều thuộc về challenger.
    - Các bản phát hành Canary có thể được kết hợp với thử nghiệm A/B để đo lường nghiêm ngặt sự khác biệt về hiệu suất.
    - Các bản phát hành Canary cũng có thể được chạy ở "chế độ YOLO", trong đó bạn quan sát sự khác biệt về hiệu suất.
    - Phiên bản khác của bản phát hành canary có thể là phát hành mô hình thách thức trước cho một thị trường nhỏ hơn, sau đó quảng bá tới tất cả các thị trường nếu mọi thứ đều ổn.
    - Nếu mô hình challenger bắt đầu gặp sự cố, hãy định tuyến lại lưu lượng truy cập đến champion.
- **Ưu điểm:**
    - Dễ hiểu.
    - Đơn giản nhất trong tất cả các chiến lược để triển khai nếu bạn đã có một số cơ sở hạ tầng gắn cờ tính năng trong tổ chức của mình.
    - Có thể sử dụng điều này với các mô hình yêu cầu tương tác của người dùng để nắm bắt hiệu suất.

- So với việc shadow deployment thì việc chạy sẽ rẻ hơn. Một suy luận cho mỗi yêu cầu.
- Nếu kết hợp với thử nghiệm A/B, nó cho phép bạn thay đổi linh hoạt lượng lưu lượng truy cập mà mỗi mô hình đang sử dụng.

- **Nhược điểm:**
  - Nó mở ra khả năng không khắt khe trong việc xác định sự khác biệt về hiệu suất.
  - Nếu việc releases không được giám sát cẩn thận, tai nạn có thể xảy ra. Đây được cho là lựa chọn kém an toàn nhất nhưng lại rất dễ bị hủy bỏ.

2.2.2.4 Interleaving Experiments

- **Triển khai:** 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 click nhiều hơn vào đề xuất từ mô hình B).
  - Nhiệm vụ đề xuất là trường hợp sử dụng điển hình của việc xen kẽ. Không phải tất cả các nhiệm vụ đều phù hợp với chiến lược này.
  - Bạn muốn tránh tạo cho mô hình một lợi thế ưu tiên người dùng không công bằng, chẳng hạn như luôn lấy dự đoán hàng đầu từ mô hình A. Có khả năng như nhau là vị trí đầu tiên đến từ mô hình A thay vì từ mô hình B. Các vị trí còn lại có thể được lấp đầy bằng cách sử dụng phương pháp soạn thảo nhóm.



Hình 16. Phương pháp soạn thảo nhóm

- **Ưu điểm:**

Netflix đã phát hiện bằng thực nghiệm rằng việc xen kẽ sẽ xác định mô hình tốt nhất một cách đáng tin cậy với kích thước mẫu nhỏ hơn đáng kể so với thử nghiệm A/B truyền thống.

Hình 17. Interleaving Experiments with Netflix

Điều này phần lớn là do cả hai mô hình đều có lưu lượng truy cập đầy đủ.

Ngược lại với việc triển khai theo dõi, chiến lược này cho phép bạn nắm bắt cách người dùng hành xử trái với dự đoán của bạn (vì dự đoán được đưa ra).

- **Nhược điểm:**
    - Việc triển khai phức tạp hơn thử nghiệm A/B.
    - Nó tăng gấp đôi sức mạnh tính toán cần thiết vì mọi yêu cầu đều nhận được dự đoán từ nhiều mô hình.
    - Nó không thể được sử dụng cho tất cả các loại nhiệm vụ. Ví dụ: nó hoạt động cho các nhiệm vụ xếp hạng/đề xuất nhưng nó không có ý nghĩa đối với các nhiệm vụ hồi quy.
    - Nó không dễ dàng mở rộng quy mô thành một số lượng lớn các mô hình thách thức. Mô hình xen kẽ 2-3 có vẻ là điểm thú vị.

2.2.2.5 Brandits

- **Triển khai:** 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.

- Brandits thêm một khái niệm khác vào quyết định sử dụng mô hình nào: chi phí cơ hội.
- Brandits không thể áp dụng cho mọi trường hợp.
- Có rất nhiều thuật toán brandits. Đơn giản nhất được gọi là epsilon - greedy. Phổ biến nhất là Thompson Samplingvà Upper Confidence Bound (UCB).

- **Ưu điểm**
  - Brandits cần ít mất dữ liệu hơn thử nghiệm A/B để xác định mô hình nào tốt hơn. Một ví dụ được đưa ra trong cuốn sách đề cập đến 630K mẫu để đạt độ tin cậy 95% khi thử nghiệm A/B VS 12K với kẻ cướp.
  - Brandits sử dụng dữ liệu hiệu quả hơn đồng thời giảm thiểu chi phí cơ hội của bạn. Trong nhiều trường hợp kẻ cướp được coi là tối ưu.
  - So với thử nghiệm A/B, kẻ cướp an toàn hơn vì nếu một mô hình thực sự tệ, thuật toán sẽ chọn nó ít thường xuyên hơn. Ngoài ra, tốc độ hội tụ sẽ nhanh hơn nên bạn có thể loại bỏ kẻ thách thức xấu một cách nhanh chóng.
- **Nhược điểm**
  - So với tất cả các chiến lược khác, kẻ cướp khó thực hiện hơn nhiều do cần phải truyền phản hồi vào thuật toán một cách liên tục.
  - Brandits chỉ có thể được sử dụng trong một số trường hợp sử dụng nhất định (xem ở trên).
  - Chúng không an toàn như Shadow Deployment vì những người thách thức chiếm lưu lượng truy cập trực tiếp.

TÀI LIỆU THAM KHẢO

1. GitHub – Continual learning and test in production.
2. VIBLO - Optimizer- Hiểu sâu về các thuật toán tối ưu (GD,SGD,Adam,..)
3. FUNDA - Gradient Descent
4. Britannica – polarity
5. Artificial Intelligence + - What is ADAGrad and How Does it Relate to Machine Learning
6. Ichi – RMSProp Học mạng thần kinh

LINK GITHUB

https://github.com/To4z/ReportFinalMachine