TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KỲ**

**NHẬP MÔN HỌC MÁY**

*Người hướng dẫn*: **THẦY LÊ ANH CƯỜNG**

*Người thực hiện*: **TRẦN VŨ – 52000732**

Lớp **: 20050281**

Khoá  **: 24**

**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**

**NHẬP MÔN HỌC MÁY**

*Người hướng dẫn*: **THẦY LÊ ANH CƯỜNG**

*Người thực hiện*: **TRẦN VŨ – 52000732**

Lớp **: 20050281**

Khoá  **: 24**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

LỜI CẢM ƠN

Em xin gửi lời cảm ơn sâu sắc và chân thành đến thầy Lê Anh Cường về sự hỗ trợ và giúp đỡ to lớn của thầy trong quá trình thực hiện bài báo cáo, thầy không chỉ là người hướng dẫn tận tâm mà qua những buổi giảng, thầy đã luôn tạo điều kiện cho chúng em thảo luận và đưa ra ý kiến, từ đó khuyến khích sự sáng tạo và tư duy. Em đã học hỏi được rất nhiều từ thầy trong quá trình học tập và thực hiện bài báo cáo này. Thầy đã cung cấp cho em những kiến thức nền tảng về mô hình học máy và các thuật toán, cũng như hướng dẫn em cách thức thực hiện bài báo cáo một cách khoa học và hiệu quả. Em cũng xin cảm ơn thầy đã giúp đỡ và hỗ trợ em để thực hiện bài báo cáo này. Đồng thời em xin cảm ơn nhà trường đã tạo điều kiện để chúng em được học tập và tìm kiếm những thông tin hữu ích liên quan đến môn học này.

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của thầy Lê Anh Cường; Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

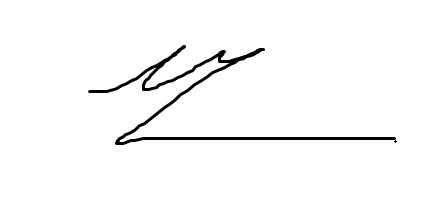
Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**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 20 tháng 12 năm 2023*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

**

*TRẦN VŨ*

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

Trong bài cáo cáo lần này tôi sẽ tìm hiểu và nguyên cứu về các phương pháp Otimizer trong mô hình học máy gồm có:

* Gradient Descent
* Stochastic Gradient Descent
* Momentum
* Adagrad
* MRSprop
* Adam

Tìm hiểu về Continual Learning và Test Production

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc153995287)

[TÓM TẮT iv](#_Toc153995288)

[MỤC LỤC 1](#_Toc153995289)

[DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT 2](#_Toc153995290)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 3](#_Toc153995291)

[CHƯƠNG 1 – CƠ SỞ LÝ THUYẾT / NGHIÊN CỨU THỰC NGHIỆM 4](#_Toc153995292)

[1.1. Optimizers là gì? 4](#_Toc153995293)

[1.1.1. Gradient Descent (GD) 5](#_Toc153995294)

[1.1.2. Stocchastic Gradient Descent (SGD) 10](#_Toc153995295)

[1.1.3. Momentum 11](#_Toc153995296)

[1.1.4. Adagrad 15](#_Toc153995297)

[1.1.5. ARMSprop (Root Mean Square Propagation) 16](#_Toc153995298)

[1.1.6. Adam 17](#_Toc153995299)

[1.2 Continual Learning và Test Production 19](#_Toc153995300)

[1.2.1 Continual Learning? (học liên tục) 19](#_Toc153995301)

[1.2.2 Test Production? 24](#_Toc153995302)

# DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

**CÁC KÝ HIỆU**

**CÁC CHỮ VIẾT TẮT**

GD Gradient Descent

SGD Stochastic Gradient Descent

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC HÌNH**

[Hình 1.1 Tổng quan về Optimizers 4](#_Toc153996466)

[Hình 1.1.2 Stocchastic Gradient Descent 10](#_Toc153996467)

[Hình 1.1.3 Momentum 12](#_Toc153996468)

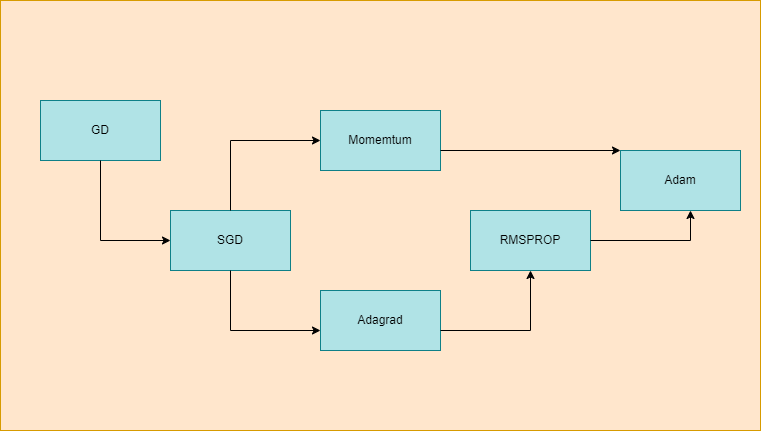
[Hình 1.1.6 Adam 18](#_Toc153996469)

CHƯƠNG 1 – CƠ SỞ LÝ THUYẾT / NGHIÊN CỨU THỰC NGHIỆM

* 1. . Optimizers là gì?

Trước khi bàn đến chi tiết, hãy tìm hiểu về khái niệm của thuật toán tối ưu (optimizers). Tại cơ bản, thuật toán tối ưu là nền tảng cho việc xây dựng mô hình neural network với mục tiêu "học" các đặc trưng (hoặc mẫu) từ dữ liệu đầu vào. Mục tiêu là tìm ra một cặp trọng số (weights) và độ chệch (bias) phù hợp để tối ưu hóa mô hình. Tuy nhiên, quá trình "học" này làm thế nào?

Không thể chỉ dựa vào việc ngẫu nhiên chọn (weights, bias) một số lần hữu hạn và hy vọng rằng ở một bước nào đó ta sẽ có lời giải. Điều này không khả thi và lãng phí tài nguyên. Để giải quyết vấn đề này, chúng ta cần một thuật toán để cải thiện từng bước trọng số và độ chệch, và đó chính là lý do tại sao các thuật toán tối ưu hóa ra đời.



Hình 1.1 Tổng quan về Optimizers

* + 1. **Gradient Descent (GD)**

Trong các bài toán tối ưu, mục tiêu thường là tìm giá trị nhỏ nhất của một hàm số. Để đạt được điều này, chúng ta thường tập trung vào việc tìm điểm mà đạo hàm của hàm số đó bằng 0. Tuy nhiên, trong thực tế, không phải lúc nào việc tính đạo hàm cũng dễ dàng. Trong trường hợp của các hàm số nhiều biến, đạo hàm có thể trở nên rất phức tạp hoặc thậm chí là không thể tính được. Do đó, thay vì giải phương trình đạo hàm bằng 0, chúng ta thường tìm đến các điểm gần cực tiểu và xem đó là nghiệm tạm thời cho bài toán.

Một phương pháp phổ biến được sử dụng để giải quyết vấn đề này là Gradient Descent, có thể được dịch sang tiếng Việt là "giảm dần độ dốc". Ý tưởng chính của phương pháp này là chọn một điểm bắt đầu ngẫu nhiên và sau mỗi vòng lặp (hoặc epoch), dịch chuyển điểm này dần về phía hướng giảm độ dốc của hàm số. Quá trình này được lặp lại cho đến khi đạt được một điểm mà giá trị của hàm số là đủ nhỏ hoặc không thay đổi đáng kể nữa.

Gradient Descent là một chiến lược hiệu quả để tiếp cận vấn đề tối ưu hóa, đặc biệt là khi không thể giải phương trình đạo hàm bằng 0 một cách tiện lợi.

Công thức: 

được sử dụng trong thuật toán Gradient Descent để điều chỉnh giá trị của biến ***x*** sau mỗi vòng lặp. Dấu trừ (-) ở đây đại diện cho việc di chuyển ngược hướng của đạo hàm. Điều này là chìa khóa để hướng mô hình về phía giá trị tối ưu (cực tiểu) của hàm số.

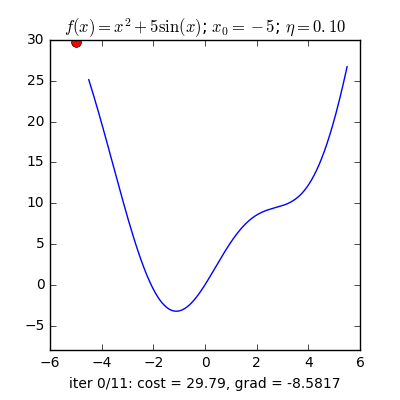
Tại sao lại ngược hướng đạo hàm? Khi ta muốn tìm giá trị nhỏ nhất của một hàm số, chúng ta quan tâm đến độ dốc của hàm tại mỗi điểm. Nếu ta di chuyển theo hướng ngược với độ dốc, tức là giảm giá trị của biến ***x*** dựa trên độ dốc tại điểm đó, chúng ta có thể tiến gần hơn tới điểm tối ưu.

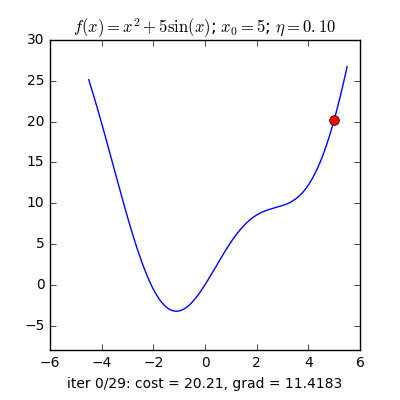
Trong ví dụ các hàm 

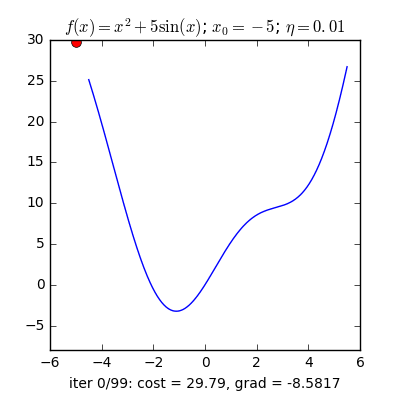
 (tức là độ dốc âm), điều này đồng nghĩa với việc hàm số đang giảm. Do đó, chúng ta muốn điều chỉnh ***x*** theo hướng tăng để tiến gần tới điểm cực tiểu. Ngược lại, khi  (tức là độ dốc dương), chúng ta muốn điều chỉnh ***x*** theo hướng giảm để tiến gần tới điểm cực tiểu.

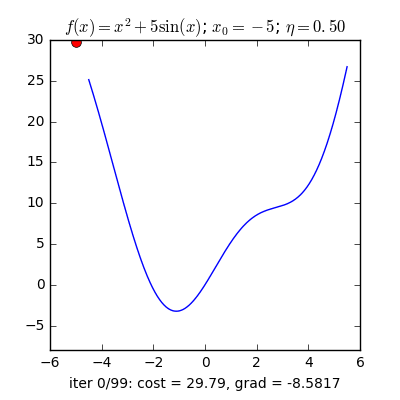
Vì vậy, ngược hướng với đạo hàm giúp chúng ta di chuyển theo hướng giảm độ dốc để đạt được giá trị tối ưu của hàm số.

1. Gradient cho hàm 1 biến



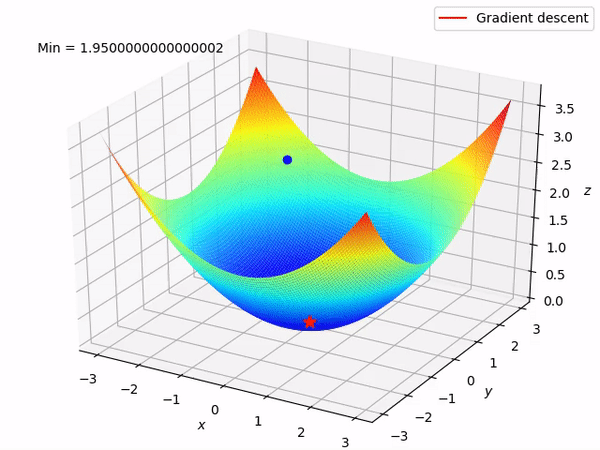






Qua những hình ảnh minh họa, ta nhận thấy rằng thuật toán Gradient Descent phụ thuộc đặc biệt vào nhiều yếu tố quan trọng, ảnh hưởng trực tiếp đến quá trình hội tụ và hiệu suất của thuật toán. Một trong những yếu tố quan trọng nhất là điểm khởi tạo (Initial Point). Việc chọn điểm khởi tạo khác nhau có thể dẫn đến kết quả cuối cùng khác nhau. Nếu chọn một điểm gần điểm cực tiểu, thuật toán có khả năng hội tụ nhanh chóng và hiệu quả hơn so với việc chọn một điểm khởi tạo xa lạ. Tốc độ học (Learning Rate) cũng là một tham số quan trọng. Nếu tốc độ học quá nhỏ, quá trình hội tụ có thể trở nên vô cùng chậm, gây ảnh hưởng tiêu cực đến quá trình huấn luyện. Ngược lại, nếu tốc độ học quá lớn, thuật toán có thể bị "quá nhảy" qua điểm cực tiểu và không thể hội tụ, dẫn đến sự không ổn định trong quá trình tối ưu hóa. Việc tinh chỉnh những yếu tố này đòi hỏi sự cân nhắc và hiểu biết sâu sắc về bản chất của hàm số và dữ liệu đang được xử lý. Quá trình này không chỉ là nghệ thuật mà còn là một phần quan trọng trong việc đạt được hiệu suất tối ưu từ thuật toán Gradient Descent trong quá trình training mô hình.

1. Gradient Descent cho hàm nhiều biến



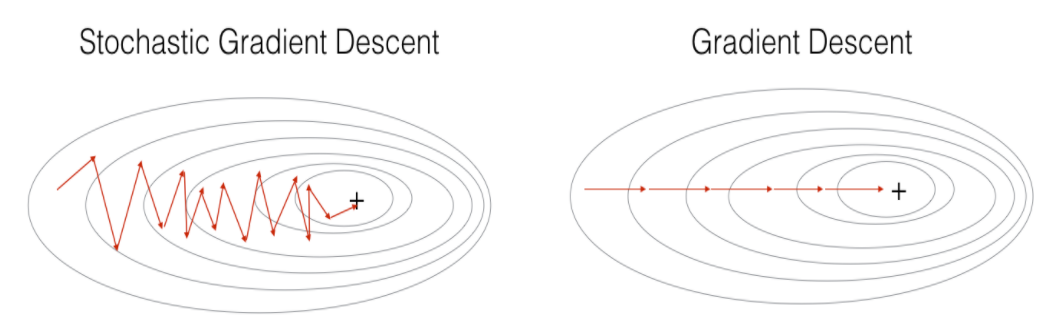
Ưu điểm:

* Thuật toán gradient descent cơ bản, dễ hiểu. Thuật toán đã giải quyết được vấn đề tối ưu model neural network bằng cách cập nhật trọng số sau mỗi vòng lặp.

Nhược điểm:

* Vì đơn giản nên thuật toán Gradient Descent còn nhiều hạn chế như phụ thuộc vào nghiệm khởi tạo ban đầu và learning rate.
* Ví dụ 1 hàm số có 2 global minimum thì tùy thuộc vào 2 điểm khởi tạo ban đầu sẽ cho ra 2 nghiệm cuối cùng khác nhau.
* Tốc độ học quá lớn sẽ khiến cho thuật toán không hội tụ, quanh quẩn bên đích vì bước nhảy quá lớn; hoặc tốc độ học nhỏ ảnh hưởng đến tốc độ training
  + 1. **Stocchastic Gradient Descent (SGD)**

Phương pháp Stochastic Gradient Descent (SGD) đại diện cho một biến thể của thuật toán Gradient Descent. Khác với việc cập nhật trọng số sau mỗi epoch như trong Gradient Descent thông thường, SGD thực hiện việc cập nhật trọng số ngay sau mỗi điểm dữ liệu, tức là N lần trong mỗi epoch với N là số lượng điểm dữ liệu.Mặc dù có thể nhìn thấy rằng SGD có thể làm giảm tốc độ của một epoch do tần suất cập nhật trọng số cao hơn, nhưng từ một góc độ khác, phương pháp này có khả năng hội tụ nhanh chóng chỉ sau vài epoch. Điều này bởi vì mỗi cập nhật trọng số dựa trên một điểm dữ liệu cụ thể, giúp nhanh chóng điều chỉnh trọng số để phản ánh thông tin mới. Công thức của SGD tương tự như Gradient Descent, nhưng thay vì tính toán trên toàn bộ tập dữ liệu, nó được thực hiện trên từng điểm dữ liệu riêng lẻ. Điều này tạo ra sự linh hoạt và hiệu suất cao, đặc biệt là trong trường hợp tập dữ liệu lớn mà không làm mất quá nhiều tài nguyên tính toán.



Hình 1.1.2 Stocchastic Gradient Descent

Nhìn vào 2 hình trên, có thể dễ dàng nhìn thấy được SGD có đường đi khá zig zac, không đi thẳng như GD vì 1 điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu. GD hạn chế với cơ sở dữ liệu lớn(vài triệu dữ liệu) vì việc tính toán đạo hàm trên toàn bộ dữ liệu qua mỗi vòng lặp trở nên cồng kềnh và không phù hợp với online learning. Khi dữ liệu cập nhập liên tục (ví dụ như thêm người đăng ký) thì mỗi phần dữ liệu ta phải tính lại đạo hàm trên toàn bộ dữ liệu dẫn đến hệ quả là thời gian tính toán lâu vì thế SGD mới ra đời giải quyết vấn đề đó, vì mỗi lần thêm dữ liệu mới vào chỉ cần cập nhật trên 1 điểm dữ liệu đó thôi và phù hợp với online learning

VD: 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:

* Thuật toán giải quyết được đối với cơ sở dữ liệu lớn mà GD không làm được. Thuật toán tối ưu này hiên nay vẫn hay được sử dụng.

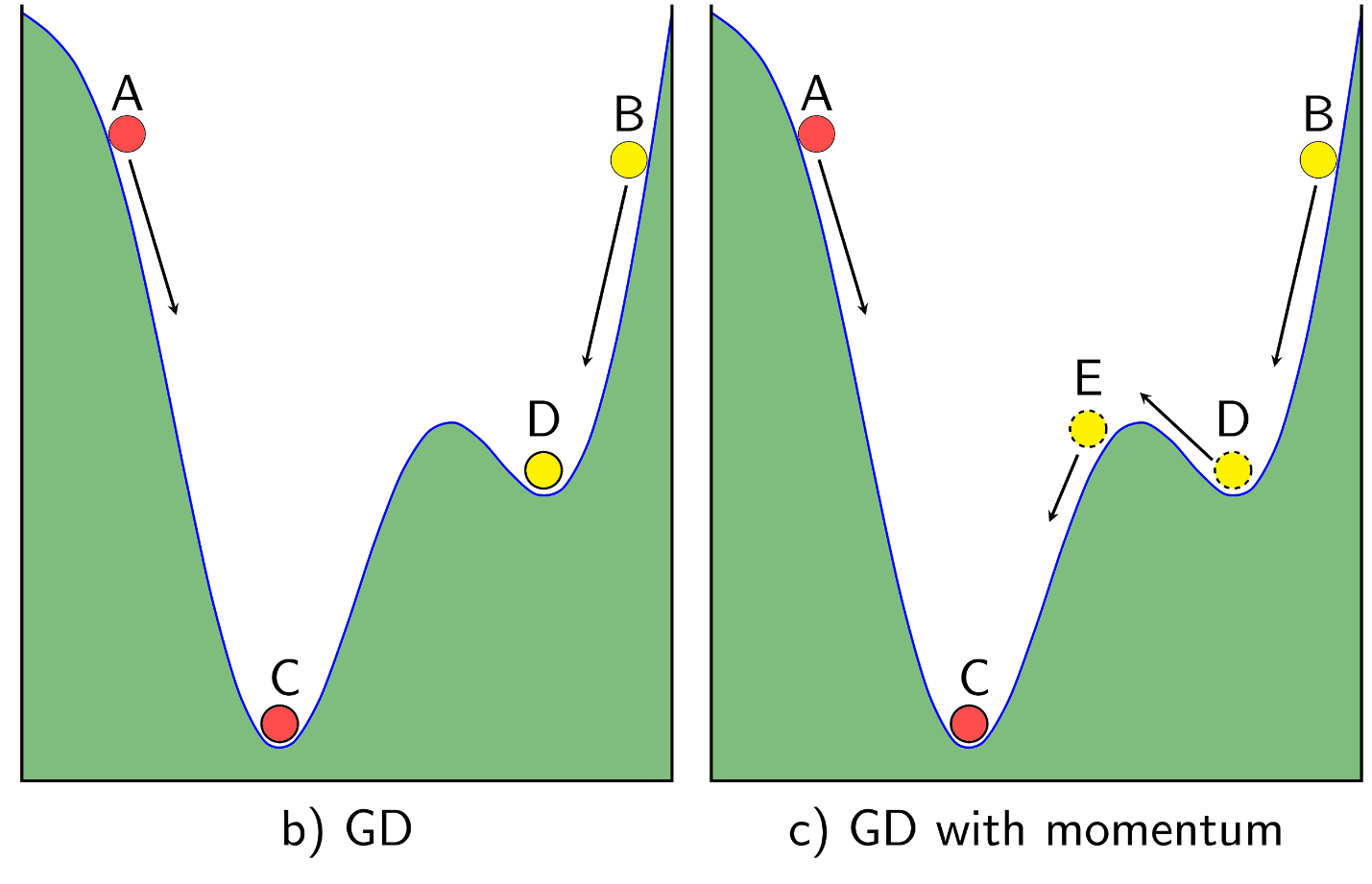
Nhược điểm:

* Thuật toán vẫn chưa giải quyết được 2 nhược điểm lớn của gradient descent (learning rate, điểm dữ liệu ban đầu).

Vì vậy ta phải kết hợp SGD với 1 số thuật toán khác như: Momentum, AdaGrad…Các thuật toán này sẽ được trình bày ở phần sau.

* + 1. **Momentum**

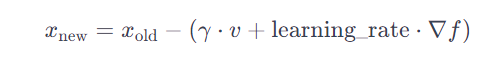
Để khắc phục các hạn chế trên của thuật toán Gradient Descent người ta dùng gradient descent with momentum. Vậy Gradient Descent with Momentum là một biến thể của thuật toán Gradient Descent, được thiết kế để khắc phục một số hạn chế của Gradient Descent thông thường. Nó sử dụng một khái niệm gọi là "momentum" để giúp tăng tốc quá trình hội tụ và tránh được những dao động không mong muốn trong quá trình tối ưu hóa.

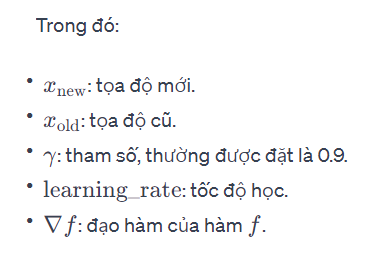


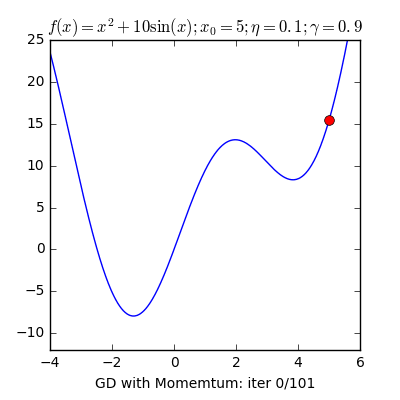
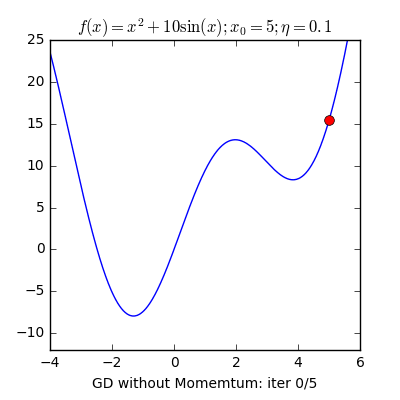
Hình 1.1.3 Momentum

Để giải thích Gradient Descent with Momentum, chúng ta có thể bắt đầu từ một quan sát đơn giản trong lĩnh vực vật lý. Nếu chúng ta thả hai viên bi từ hai điểm khác nhau A và B, chúng sẽ cuốn xuống hai điểm C và D tương ứng. Tuy nhiên, không mong muốn là viên bi thứ hai sẽ dừng lại ở điểm D (cực tiểu địa phương), mà chúng ta muốn nó tiếp tục cuộn tới điểm C (cực tiểu toàn cục). Để thực hiện điều này, ta cần cung cấp cho viên bi thứ hai một vận tốc ban đầu đủ lớn để vượt qua điểm E và tiếp tục đến điểm C. Ý tưởng này đã tạo ra thuật toán Gradient Descent with Momentum, hay nói cách khác, theo đà tiến tới.

Dưới góc độ toán học, thuật toán Momentum có công thức cập nhật như sau:







Qua ví dụ minh hoạ của hàm  
 

chúng ta có thể thấy rõ sự ảnh hưởng của Gradient Descent with Momentum so với Gradient Descent without Momentum.

Trong trường hợp của Gradient Descent without Momentum, thuật toán hội tụ sau khoảng 5 vòng lặp nhưng lại không đạt được global minimum. Điều này cho thấy nhược điểm của thuật toán khi nó dễ mắc kẹt ở các điểm cực tiểu địa phương mà không thể tiến tới cực tiểu toàn cục.

Ngược lại, Gradient Descent with Momentum mất nhiều vòng lặp hơn nhưng cuối cùng nó đạt được nghiệm tiến tới global minimum. Hình ảnh cho thấy thuật toán này có khả năng "vượt qua" các điểm cực tiểu địa phương và đồng thời giảm độ chói lọi, dao động qua lại quanh điểm global minimum trước khi dừng lại. Điều này làm nổi bật sức mạnh của Momentum trong việc giúp thuật toán vượt qua các thách thức và tối ưu hóa hiệu suất học của mô hình.

Ưu điểm:

* Thuật toán tối ưu giải quyết được vấn đề: Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum.

Nhược điểm:

* Tuy momentum giúp hòn bi vượt dốc tiến tới điểm đích, tuy nhiên khi tới gần đích, nó vẫn mất khá nhiều thời gian giao động qua lại trước khi dừng hẳn, điều này được giải thích vì viên bi có đà.
  + 1. **Adagrad**

Không giống như các thuật toán trước đó thì learning rate hầu như giống nhau trong quá trình training (learning rate là hằng số), Adagrad coi learning rate là 1 tham số. Tức là Adagrad sẽ cho learning rate biến thiên sau mỗi thời điểm t. Adagrad là một thuật toán tối ưu hóa được thiết kế để điều chỉnh tốc độ học (learning rate) của mỗi tham số trong mô hình theo cách tự động. Nó được thiết kế đặc biệt để xử lý vấn đề của việc chọn tốc độ học, một thách thức quan trọng trong quá trình huấn luyện mô hình máy học.

Ý tưởng cơ bản của Adagrad là điều chỉnh tốc độ học của từng tham số dựa trên lịch sử của các đạo hàm trước đó. Nếu một tham số có đạo hàm lớn, có thể nói rằng nó quan trọng và nên có tốc độ học nhỏ hơn để tránh việc overshooting. Ngược lại, nếu đạo hàm nhỏ, tốc độ học có thể được tăng lên để học nhanh hơn. Điều này giúp cân bằng tốc độ học cho từng tham số dựa trên đặc điểm của chúng.

Công thức:



Trong đó:

n: hằng số

gt: gradient tại thời điểm t

ϵ: hệ số tránh lỗi (chia cho mẫu bằng 0)

G: là ma trận chéo mà mỗi phần tử trên đường chéo (i,i) là bình phương của đạo hàm vectơ tham số tại thời điểm t.

Ưu điểm:

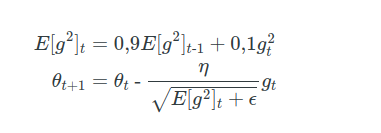
* Một lơi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, chỉ cần để tốc độ học default là 0.01 thì thuật toán sẽ tự động điều chỉnh.

Nhược điểm:

* Yếu điểm của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm tốc độ học cực kì nhỏ, làm việc training trở nên đóng băng.
  + 1. **ARMSprop (Root Mean Square Propagation)**

Là một thuật toán tối ưu hóa được thiết kế để giải quyết vấn đề của Adagrad, đặc biệt là vấn đề tỷ lệ học giảm dần quá nhanh. Adagrad là một thuật toán tối ưu hóa tỷ lệ học động cho từng tham số dựa trên lịch sử của gradient. Tuy nhiên, nó có thể dẫn đến việc tỷ lệ học giảm quá nhanh, đặc biệt là khi sử dụng trong các bài toán deep learning.

RMSprop giải quyết vấn đề này bằng cách thay đổi cách tính toán đối với lịch sử gradient. Thay vì tích lũy toàn bộ lịch sử bình phương gradient như Adagrad, RMSprop sử dụng trung bình có trọng số (exponential moving average) của bình phương gradient.



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:

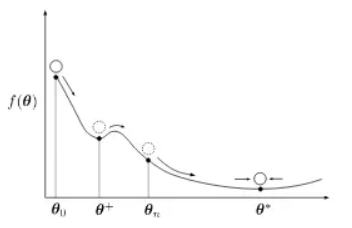
* Thuật toán RMSprop có thể cho kết quả nghiệm chỉ là local minimum chứ không đạt được global minimum như Momentum.

Vì vậy ta sẽ kết hợp cả 2 thuật toán Momentum với RMSprop cho ra 1 thuật toán tối ưu Adam

* + 1. **Adam**

Adam, giống với Adagradvà RMSprop, là một thuật toán tối ưu hóa gradient descent được thiết kế để cập nhật trọng số của mô hình neural network trong quá trình đào tạo. Trong quá trình này, Adam duy trì trung bình bình phương độ dốc (vt) và đồng thời duy trì trung bình độ dốc quá khứ (mt), giống với cách làm của RMSprop và Adagrad. Nó cũng chứa trong mình tính chất của động lượng (momentum), nơi nó tích tụ động lượng từ gradient để vượt qua các điểm thấp và tiến về các điểm thấp hơn.

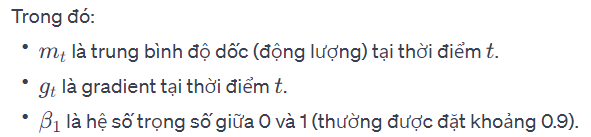
Một cách mô tả vật lý cho Adam là nhìn nhận nó như một quả cầu rất nặng và có ma sát. Trong khi động lượng giống như quả cầu lao xuống dốc, Adam như một quả cầu nặng và chịu sự ảnh hưởng của ma sát. Điều này giúp nó dễ dàng vượt qua các điểm cực tiểu cục bộ và đạt tới điểm tối ưu nhất, tạo ra một hiệu ứng được gọi là "Heavy Ball with Friction" (HBF). Công thức tính toán hệ số  mt trong Adam chính là yếu tố quyết định sự ảnh hưởng của động lượng và ma sát, giúp thuật toán điều chỉnh tỷ lệ học một cách linh hoạt và hiệu quả.

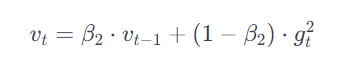
****

Hình 1.1.6 Adam

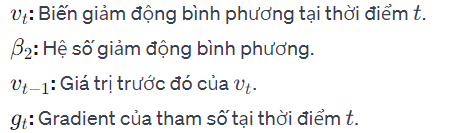
**Công thức update:**

****



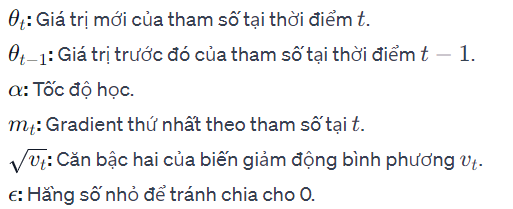


Trong đó:





Trong đó:



## 1.2 Continual Learning và Test Production

### 1.2.1 Continual Learning? (học liên tục)

Học tập liên tục là ý tưởng cập nhật mô hình khi có dữ liệu mới, điều này giúp mô hình theo kịp sự phân bổ dữ liệu hiện tại.

Sau khi mô hình được cập nhật, mô hình đó không thể được phát hành một cách sai lầm vào sản xuất. Nó cần phải được thử nghiệm để đảm bảo rằng nó an toàn và tốt hơn so với mô hình hiện tại đang được sản xuất. Đây là phần tiếp theo " Test Production".

#### Học tập liên tục thường bị hiểu sai:

Học tập liên tục không đề cập đến một lớp thuật toán ML đặc biệt cho phép cập nhật mô hình dần dần khi có mọi điểm dữ liệu mới.

Ví dụ về lớp thuật toán đặc biệt này là cập nhật bayes tuần tự và Bộ phân loại KNN. Lớp thuật toán này có quy mô nhỏ và đôi khi được gọi là "thuật toán học trực tuyến".

Khái niệm Học tập liên tục có thể được áp dụng cho bất kỳ thuật toán ML được giám sát nào. Không, chỉ là một lớp học đặc biệt thôi.

Học tập liên tục không có nghĩa là bắt đầu công việc đào tạo lại mỗi khi có mẫu dữ liệu mới. Trên thực tế, điều này rất nguy hiểm vì nó khiến mạng lưới thần kinh dễ bị lãng quên.

#### **Tại sao phải học liên tục?**

Lý do cơ bản là để giúp mô hình theo kịp sự thây đổi phân phối dữ liệu. Có một số trường hợp sử dụng trong đó việc thích ứng nhanh chóng với việc thay đổi phân phối là rất quan trọng. Dưới đây là một số ví dụ:

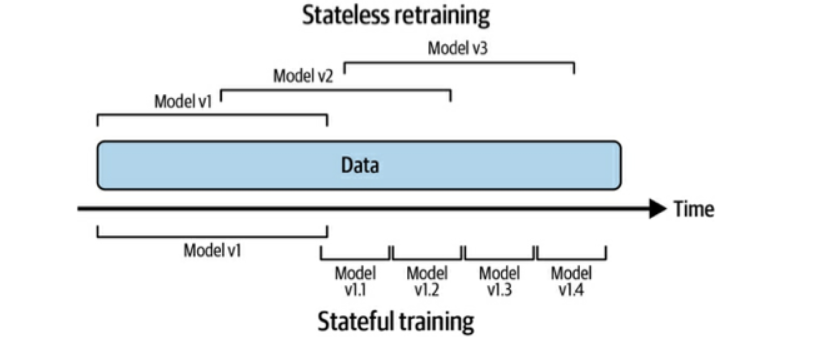
#### **Các trường hợp sử dụng có thể xảy ra những thay đổi nhanh chóng và bất ngờ**

Giả sử có một sự kiện thể thao lớn như một trận đấu bóng đá quan trọng diễn ra vào một buổi tối cuối tuần. Trong khoảng thời gian này, có thể xuất hiện một lượng lớn người xem, và nhiều người có thể quyết định đặt đơn hàng đồ ăn để thưởng thức sự kiện.

#### **Các trường hợp sử dụng không thể lấy dữ liệu đào tạo cho một sự kiện cụ thể**.

Một ví dụ cho điều này là các mô hình thương mại điện tử trong dịp Black Friday hay một số sự kiện sale khác chưa từng được thử trước đây. Rất khó để thu thập dữ liệu lịch sử để dự đoán hành vi của người dùng trong Black Friday, vì vậy mô hình của bạn phải thích ứng suốt cả ngày hôm đó.

#### **Đào tạo lại không trạng thái và Đào tạo có trạng thái**

****

#### **Đào tạo lại không trạng thái**

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 tập liên tục bằng cách sử dụng đào tạo lại không trạng thái.

#### **Đào tạo có trạng thái**

Đào tạo có trạng thái, hay còn gọi là tinh chỉnh mô hình, là một phương pháp trong học máy cho phép cập nhật mô hình với dữ liệu mới một cách hiệu quả và tiết kiệm tài nguyên tính toán.

#### **Khởi Tạo Mô Hình và Tiếp Tục Huấn Luyện:**

* Bắt đầu với một mô hình được huấn luyện từ trước, thường là sử dụng trọng số từ một mô hình đã được đào tạo trước đó (pre-trained model).
* Tiếp tục huấn luyện mô hình với dữ liệu mới, thường là một lượng dữ liệu ít hơn so với quá trình đào tạo ban đầu.

#### **Ưu Điểm của Đào Tạo Có Trạng Thái:**

* Cập nhật mô hình với lượng dữ liệu ít hơn đáng kể, giảm tải tài nguyên tính toán.
* Hội tụ nhanh hơn và sử dụng ít năng lượng tính toán hơn so với việc đào tạo lại từ đầu.

#### **Lặp Lại Mô Hình và Dữ Liệu:**

* Đào tạo có trạng thái thường được sử dụng để lặp lại dữ liệu, giúp kết hợp dữ liệu mới vào kiến trúc mô hình hiện có.

#### **Những thách thức học tập liên tục**

Học tập liên tục đã được áp dụng và đạt nhiều thành công lớn. Tuy nhiên, có ba thách thức lớn cần phải vượt qua.

#### **Thử thách truy cập dữ liệu mới**

Cập nhật mô hình liên tục. Dữ liệu đào tạo chất lượng được gắn nhãn mỗi giờ. Nhịp độ cập nhật càng ngắn thì thách thức này càng trở nên nghiêm trọng.

#### **Bốn giai đoạn của việc học liên tục**

**Giai đoạn 1:** Đào tạo lại thủ công, không trạng thái

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

* hiệu suất của mô hình đã suy giảm đến mức hiện tại nó gây hại nhiều hơn là có lợi
* có thời gian để cập nhật mô hình.

**Giai đoạn 2**: Đào tạo lại 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. Ở lại giai đoạn 1 đã trở thành một nỗi đau quá lớn không thể bỏ qua.

Tần suất đào tạo lại ở giai đoạn này thường dựa trên "gut feeling".

Điểm giữa giai đoạn 1 và giai đoạn 2 thường là một tập lệnh do ai đó viết 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.

**Giai đoạn 3**: Đào tạo trạng thái tự động theo lịch trình cố định

Để đạt được điều này, cần phải cấu hình lại tập lệnh và cách theo dõi dòng dữ liệu cũng như mô hình.

**Giai đoạn 4**: Học tập liên tục

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:

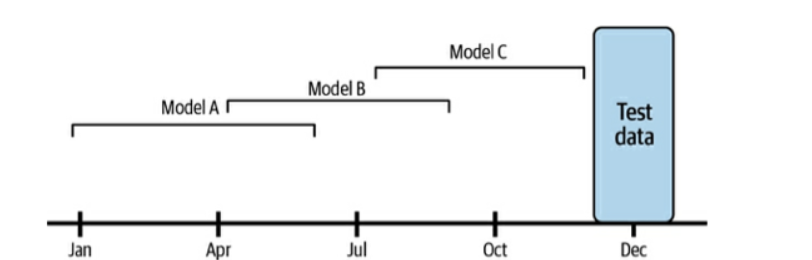
* Dựa trên thời gian
* Dựa trên hiệu suất: ví dụ hiệu suất đã giảm xuống dưới x%.
* Không phải lúc nào cũng có thể đo lường trực tiếp độ chính xác trong quá trình sản xuất, vì vậy có thể cần sử dụng một số proxy yếu hơn.
* Dựa trên khối lượng: Tổng lượng dữ liệu được dán nhãn tăng 5%
* Dựa trên sự trôi dạt: ví dụ: khi phát hiện thấy sự thay đổi phân phối dữ liệu "chính".

Điều khó khăn khi sử dụng trình kích hoạt dựa trên sự trôi dạt là, như đã đề cập trong chương trước, các dịch chuyển phân phối dữ liệu chỉ là vấn đề nếu chúng làm cho hiệu suất mô hình của bạn suy giảm. Có thể khó để biết khi nào điều đó xảy ra.

#### **Đo lường giá trị của độ mới dữ liệu**

Một cách để định lượng giá trị của dữ liệu mới hơn là huấn luyện cùng một kiến ​​trúc mô hình với dữ liệu từ 3 khoảng thời gian khác nhau, sau đó kiểm tra từng mô hình dựa trên dữ liệu được gắn nhãn hiện tại.

Nếu bạn phát hiện ra rằng việc để mô hình cũ 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 cần đào tạo lại sau đó.



#### **Khi nên nào nên lập lại mô hình**

* Nếu tiếp tục giảm kích hoạt đào tạo lại việc lặp lại dữ liệu và không thu được nhiều lợi ích, nên đầu tư vào việc 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 tăng hiệu suất 1% với sức mạnh tính toán 1X, hãy ưu tiên việc lặp lại dữ liệu hơn việc lặp lại mô hình.

### 1.2.2 Test Production?

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

#### **Đánh giá ngoại tuyến trước khi triển khai**

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

* Phần tách thử nghiệm thường ở dạng tĩnh để 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).

#### **Thử nghiệm A/B**

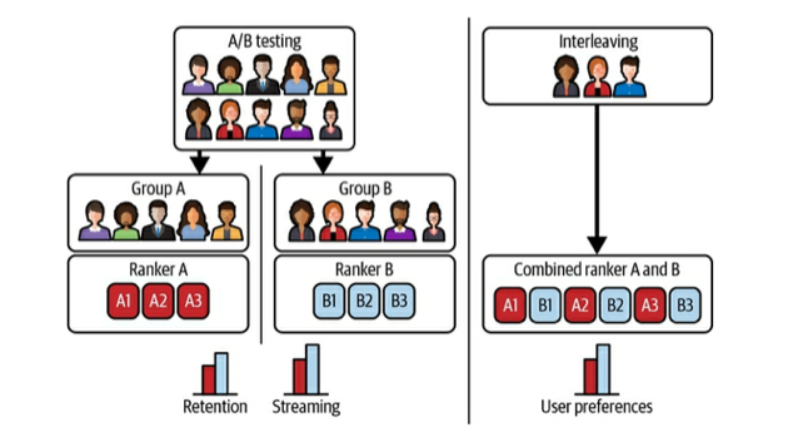
Thử nghiệm A/B là một phương pháp thử nghiệm thường được sử dụng trong lĩnh vực tiếp thị và nghiên cứu người dùng để so sánh hiệu suất giữa hai phiên bản khác nhau của một yếu tố cụ thể. Trong ngữ cảnh kiểm thử mô hình hoặc sản phẩm, thử nghiệm A/B thường được sử dụng để đánh giá hiệu suất giữa hai mô hình hoặc hai phiên bản của một sản phẩm.

#### **Thí nghiệm xen kẽ**

Trực giác: 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, 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.

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.



TÀI LIỆU THAM KHẢO

**Tiếng Việt**

Optimizer là gì? [xem tại đây](https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8)

**Tiếng Anh**

Continual Learning và Test Production ? [xem tại đây](https://github.com/serodriguez68/designing-ml-systems-summary/blob/main/09-continual-learning-and-test-in-production.md)