**Trần Trung Hiếu\_52000888  
Đề bài:**

Trình bày một bài nghiên cứu, đánh giá của em về các vấn đề sau:

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

**Bài làm:**

**Câu 1:** Thuật toán tối ưu (optimizers).Về cơ bản, thuật toán tối ưu là cơ sở để xây dựng mô hình neural network với mục đích "học " được các features (hay pattern) của dữ liệu đầu vào, từ đó có thể tìm 1 cặp weights và bias phù hợp để tối ưu hóa model. Nhưng vấn đề là "học" như thế nào? Cụ thể là weights và bias được tìm như thế nào! Đâu phải chỉ cần random (weights, bias) 1 số lần hữu hạn và hy vọng ở 1 bước nào đó ta có thể tìm được lời giải. Rõ ràng là không khả thi và lãng phí tài nguyên. Chúng ta phải tìm 1 thuật toán để cải thiện weight và bias theo từng bước, và đó là lý do các thuật toán optimizer ra đời.

Một vài các yếu tố hay được sử dụng để đánh giá một thuật toán optimizer:

* Hội tụ nhanh (trong quá trình train)
* Sự tổng quát hóa cao (vẫn nhận dạng được những mẫu chưa từng được huấn luyện)
* Độ chính xác cao.

**Các thuật toán Optimizers phổ biến:  
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 đâu 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 dịch ra tiếng Việt là giảm dần độ dốc, nên 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.

*Công thức : xnew = xold - learningrate.gradient(x)*

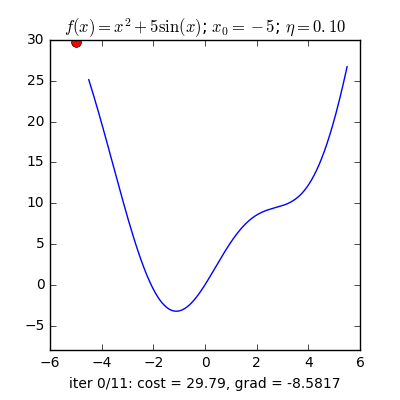
Đặt câu hỏi tại sao có công thức đó ? Công thức trên được xây dựng để cập nhật lại nghiệm sau mỗi vòng lặp . Dấu '-' trừ ở đây ám chỉ ngược hướng đạo hàm. Đặt tiếp câu hỏi tại sao lại ngược hướng đạo hàm ?

Ví dụ như đối với hàm f(x)= 2x +5sin(x) như hình dưới thì f'(x) =2x + 5cos(x)

với x\_old =-4 thì f'(-4) <0 => x\_new > x\_old nên nghiệm sẽ di chuyển về bên phải tiến gần tới điểm cực tiểu.

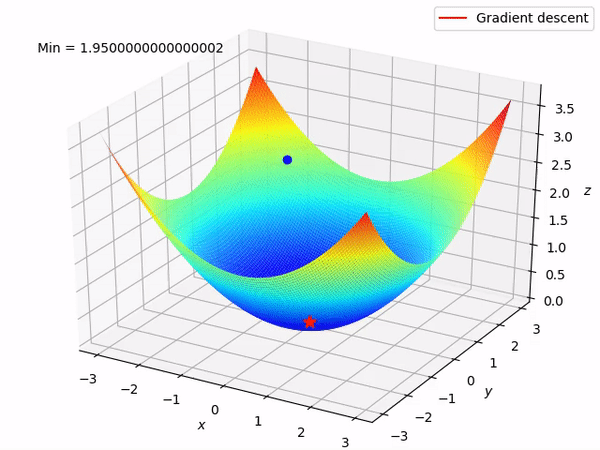
ngược lại với x\_old =4 thì f'(4) >0 => x\_new <x\_old nên nghiệm sẽ di chuyển về bên trái tiến gần tới điểm cực tiểu.

1. Gradient cho hàm 1 biến :



Qua các hình trên ta thấy Gradient descent phụ thuộc vào nhiều yếu tố : như nếu chọn điểm x ban đầu khác nhau sẽ ảnh hưởng đến quá trình hội tụ; hoặc tốc độ học (learning rate) quá lớn hoặc quá nhỏ cũng ảnh hưởng: nếu tốc độ học quá nhỏ thì tốc độ hội tụ rất chậm ảnh hưởng đến quá trình training, còn tốc độ học quá lớn thì tiến nhanh tới đích sau vài vòng lặp tuy nhiên thuật toán không hội tụ, quanh quẩn quanh đích vì bước nhảy quá lớn.

1. Gradient descent cho hàm nhiều biến :



**Ưu điểm :**

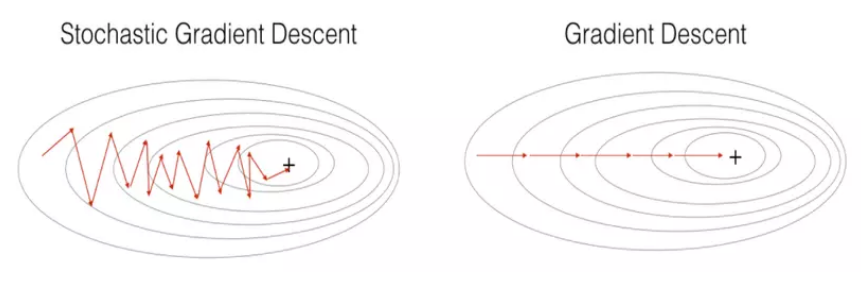
* Thuật toán gradient descent cơ bản, dễ hiểu.
* Thuật toán đã giải quyết được vấn đề tối ưu model neural network bằng cách cập nhật trọng số sau mỗi vòng lặp.

**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

**2. Stochastic Gradient Descent (SGD).**

Stochastic là 1 biến thể của Gradient Descent . Thay vì sau mỗi epoch chúng ta sẽ cập nhật trọng số (Weight) 1 lần thì trong mỗi epoch có N điểm dữ liệu chúng ta sẽ cập nhật trọng số N lần. Nhìn vào 1 mặt , SGD sẽ làm giảm đi tốc độ của 1 epoch. Tuy nhiên nhìn theo 1 hướng khác,SGD sẽ hội tụ rất nhanh chỉ sau vài epoch. Công thức SGD cũng tương tự như GD nhưng thực hiện trên từng điểm dữ liệu.



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. Đặt câu hỏi tại sao phải dùng SGD thay cho GD mặt dù đường đi của nó khá zig zắc ? 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. Vậy online learning là gì? 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 => 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 :**

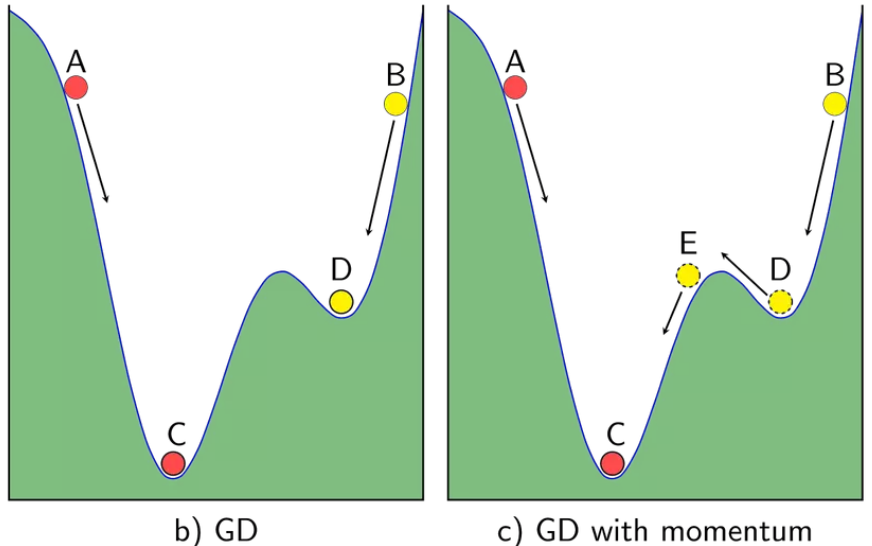
* Thuật toán giải quyết được đối với cơ sở dữ liệu lớn mà GD không làm được. Thuật toán tối ưu này hiên nay vẫn hay được sử dụng.

**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.

**3. Momentum**

Để khắc phục các hạn chế trên của thuật toán Gradient Descent người ta dùng gradient descent with momentum. Vậy gradient with momentum là gì ?



Để giải thích được Gradient with Momentum thì trước tiên ta nên nhìn dưới góc độ vật lí: Như hình b phía trên, nếu ta thả 2 viên bi tại 2 điểm khác nhau A và B thì viên bị A sẽ trượt xuống điểm C còn viên bi B sẽ trượt xuống điểm D, nhưng ta lại không mong muốn viên bi B sẽ dừng ở điểm D (local minimum) mà sẽ tiếp tục lăn tới điểm C (global minimum). Để thực hiện được điều đó ta phải cấp cho viên bi B 1 vận tốc ban đầu đủ lớn để nó có thể vượt qua điểm E tới điểm C. Dựa vào ý tưởng này người ta xây dựng nên thuật toán Momentum ( tức là theo đà tiến tới ).

Nhìn dưới góc độ toán học, ta có công thức Momentum:

xnew = xold -(gama.v + learningrate.gradient)

Trong đó :

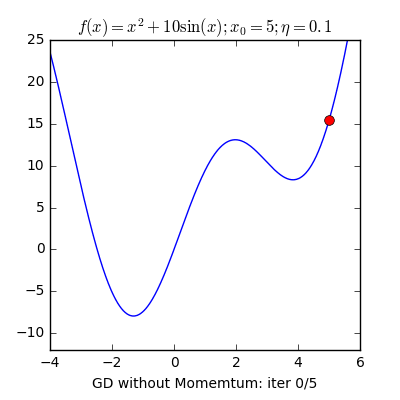
xnew: tọa độ mới

xod : tọa độ cũ

gama: parameter , thường =0.9

learningrate : tốc độ học

gradient : đạo hàm của hàm f



Qua 2 ví dụ minh họa trên của hàm f(x) = x.2 + 10sin(x), ta thấy GD without momentum sẽ hội tụ sau 5 vòng lặp nhưng không phải là global minimum. Nhưng GD with momentum dù mất nhiều vòng lặp nhưng nghiệm tiến tới global minimum, qua hình ta thấy nó sẽ vượt tốc tiến tới điểm global minimum và dao động qua lại quanh điểm đó trước khai dừng lại.

**Ưu điểm :**

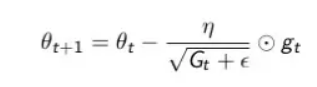
* Thuật toán tối ưu giải quyết được vấn đề: Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum.

**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.

**4. 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.



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 :**

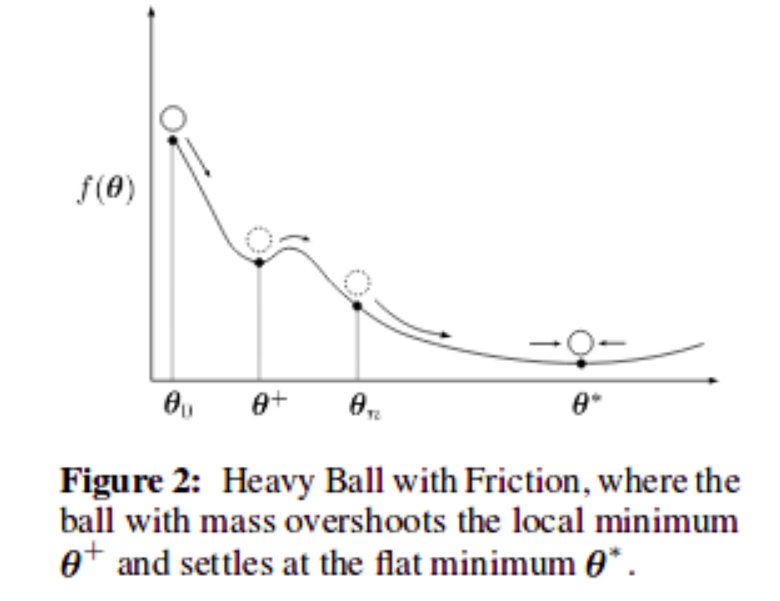
* Một lơi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, chỉ cần để tốc độ học default là 0.01 thì thuật toán sẽ tự động điều chỉnh.

**Nhược điểm :**

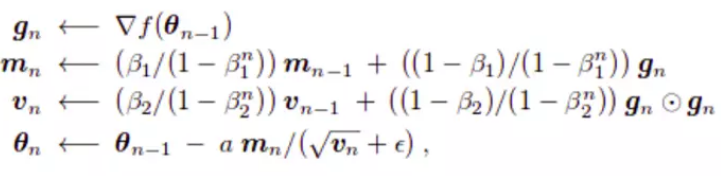
* Yếu điểm của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm tốc độ học cực kì nhỏ, làm việc training trở nên đóng băng.

**5. Adaptive Moment Estimation(Adam)**

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.



Công thức :



Ưu điểm

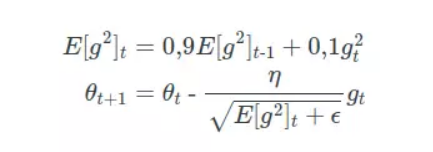
* Phương pháp này quá nhanh và hội tụ nhanh chóng.
* Khắc phục tỷ lệ học tập biến mất, phương sai cao.

Nhược điểm:

* Tốn kém về mặt tính toán.

**6. RMSprop**

RMSprop giải quyết vấn đề tỷ lệ học giảm dần của Adagrad bằng cách chia tỷ lệ học cho trung bình của bình phương gradient.



**Ưu điểm :**

* Ưu điểm rõ nhất của RMSprop là giải quyết được vấn đề tốc độ học giảm dần của Adagrad ( vấn đề tốc độ học giảm dần theo thời gian sẽ khiến việc training chậm dần, có thể dẫn tới bị đóng băng )

**Nhược điểm :**

* Thuật toán RMSprop có thể cho kết quả nghiệm chỉ là local minimum chứ không đạt được global minimum như Momentum. Vì vậy người ta sẽ kết hợp cả 2 thuật toán Momentum với RMSprop cho ra 1 thuật toán tối ưu Adam. Chúng ta sẽ trình bày nó trong phần sau.

**Kết luận:**

Nhưng theo những gì tìm hiểu, thuật toán tối ưu hóa tốt nhất cho neural network là Adaptive Moment Estimation (Adam) . Thuật toán tối ưu hóa này hoạt động rất tốt cho hầu hết mọi vấn đề về deep learning mà gặp phải.

Đó là lý do tại sao Adam Optimizer là thuật toán tối ưu hóa mặc định cho mọi vấn đề mình muốn giải quyết. Chỉ trong một số rất ít trường hợp ta mới chuyển sang các thuật toán tối ưu hóa khác mà đã giới thiệu trước đó.Nếu ai đó muốn đào tạo neural network trong thời gian ngắn hơn và hiệu quả hơn thì Adam là lựa chọn ưu tiên nhất.

**Câu 2: Continual learning và Test production**

**2.1. Continual Learning**

**2.1.1. Khái niệm**

Continual Learning là ý tưởng cập nhật mô hình của bạn khi có dữ liệu mới; điều này làm cho mô hình của bạn theo kịp các phân phối dữ liệu hiện tại.

Học tập liên tục thường bị hiểu sai:

* Học 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 bayesian tuần tự và phân loại KNN . Lớp thuật toán này nhỏ và đôi khi được gọi là "thuật toán học trực tuyến".

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

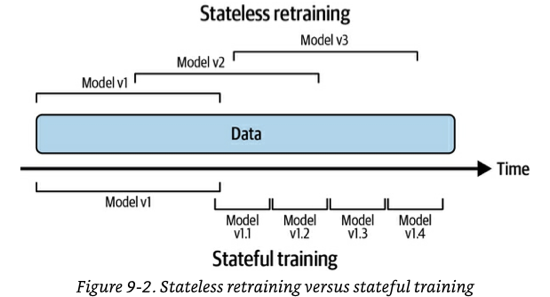
* Học liên tục KHÔNG có nghĩa là bắt đầu công việc đào tạo lại mỗi khi có mẫu dữ liệu mới. Trên thực tế, điều này rất nguy hiểm vì nó khiến mạng lưới thần kinh dễ bị lãng quên một cách thảm khốc.
* Hầu hết các công ty sử dụng phương pháp học tập liên tục đều cập nhật mô hình của họ theo từng đợt nhỏ (ví dụ cứ sau 512 hoặc 1024 ví dụ). Số lượng ví dụ tối ưu phụ thuộc vào nhiệm vụ.

Lý do cơ bản cho điều này là để giúp mô hình của bạn đồng bộ với sự thay đổi trong phân phối dữ liệu. Có một số trường hợp sử dụng trong đó việc thích ứng nhanh chóng với sự thay đổi phân phối là quan trọng.

Dưới đây là một số ví dụ:

* Trường hợp sử dụng trong đó có thể xảy ra sự thay đổi đột ngột và không ngờ: Các trường hợp sử dụng như chia sẻ chuyến đi đang chịu ảnh hưởng từ điều này. Ví dụ, có thể có một buổi hòa nhạc ở một khu vực ngẫu nhiên vào một ngày Thứ Hai ngẫu nhiên, và "mô hình ML định giá Thứ Hai" có thể không được trang bị đầy đủ để xử lý điều này.
* Trường hợp sử dụng trong đó không thể có dữ liệu đào tạo cho một sự kiện cụ thể. Một ví dụ về điều này là mô hình thương mại điện tử vào Ngày Mua Sắm Giảm Giá (Black Friday) hoặc một sự kiện giảm giá khác chưa từng được thử nghiệm trước đó. Rất khó để thu thập dữ liệu lịch sử để dự đoán hành vi người dùng vào Ngày Mua Sắm Giảm Giá, nên mô hình của bạn phải thích ứng qua cả ngày.
* Trường hợp sử dụng 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) không có dữ liệu lịch sử (hoặc dữ liệu đã lỗi thời). Nếu bạn không điều chỉnh mô hình ngay khi có dữ liệu từ người dùng đó, bạn sẽ không thể đề xuất những điều liên quan cho người dùng đó.

**2.1.2. Stateless retraining và Stateful training.**



Huấn luyện không trạng thái:

Huấn luyện 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 sự 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 học liên tục bằng cách sử dụng huấn luyện không trạng thái.

Huấn luyện có trạng thái (còn được gọi là tinh chỉnh, học tăng cường)

Khởi tạo mô hình của bạn với trọng số từ vòng huấn luyện trước đó và tiếp tục huấn luyện bằng cách sử dụng dữ liệu mới chưa được nhìn thấy.

* Cho phép mô hình của bạn cập nhật với lượng dữ liệu đáng kể ít hơn.
* Cho phép mô hình của bạn hội tụ nhanh hơn và sử dụng ít công suất tính toán hơn.
* Một số công ty báo cáo giảm 45% công suất tính toán.
* Lý thuyết làm cho nó có thể tránh hoàn toàn việc lưu trữ dữ liệu sau khi dữ liệu đã được sử dụng cho việc huấn luyện (và để lại một khoảng thời gian an toàn). Điều này lý thuyết loại bỏ mối quan ngại về quyền riêng tư dữ liệu.
* Trong thực tế, hầu hết các công ty thường thực hành "hãy theo dõi tất cả mọi thứ" và do dự về việc vứt bỏ dữ liệu ngay cả khi không cần thiết nữa.
* Mỗi khi cần, bạn sẽ cần chạy huấn luyện không trạng thái với một lượng lớn dữ liệu để hiệu chỉnh lại mô hình.
* Khi cơ sở hạ tầng của bạn được thiết lập đúng, việc chuyển từ huấn luyện không trạng thái sang huấn luyện có trạng thái trở thành việc nhấn một nút.
* Lặp mô hình so với lặp dữ liệu: Huấn luyện có trạng thái chủ yếu được sử dụng để tích hợp dữ liệu mới vào một kiến trúc mô hình hiện tại và cố định (tức là lặp dữ liệu). Nếu bạn muốn thay đổi đặc điểm hoặc kiến trúc của mô hình của mình, bạn sẽ cần thực hiện một vòng lặp đầu tiên của huấn luyện không trạng thái.
* Có một số nghiên cứu về cách chuyển trọng số từ một kiến trúc mô hình sang một kiến trúc mới (chuyển giao kiến thức Net2Net, phẫu thuật mô hình). Tuy nhiên, hiện chưa có hoặc ít sự áp dụng của các kỹ thuật này trong ngành công nghiệp.

**2.1.3. Các thách thức của Continual Learning.**

Continual Learning đã được áp dụng trong nhiều lĩnh vực với thành công lớn. Tuy nhiên, nó có các vấn đề cần phải vượt qua.

* Thách thức về việc truy cập dữ liệu mới
* Thách thức đánh giá
* Thách thức về tỷ lệ dữ liệu
* Thách thức về thuật toán

**2.1.4. Các giai đoạn của Continual Learning.**

**Giai đoạn 1: Huấn luyện thủ công không trạng thái**

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

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

Giai đoạn này thường xảy ra khi các mô hình chính của một lĩnh vực đã được phát triển và do đó ưu tiên của bạn không còn là tạo ra mô hình mới, mà là duy trì và cải thiện những mô hình hiện tại. Việc ở lại ở giai đoạn 1 đã trở thành một vấn đề quá lớn để phớt lờ.

Tần suất huấn luyện lại ở giai đoạn này thường dựa trên "cảm giác".

Điểm chuyển động giữa giai đoạn 1 và giai đoạn 2 thường là một đoạn mã mà ai đó viết để chạy huấn luyện không trạng thái định kỳ. Việc viết mã này có thể rất dễ hoặc rất khó tùy thuộc vào số lượng phụ thuộc cần được đồng bộ để huấn luyện lại một mô hình.

Các bước cấp cao của đoạn mã này là:

1. Rút dữ liệu.
2. Giảm mẫu hoặc tăng mẫu nếu cần.
3. Trích đặc trưng.
4. Xử lý và/hoặc chú thích nhãn để tạo dữ liệu đào tạo.
5. Bắt đầu quá trình huấn luyện.
6. Đánh giá mô hình mới.
7. Triển khai nó.

Có 2 phần cơ sở hạ tầng bổ sung bạn sẽ cần triển khai cho đoạn mã:

Một lịch trình

Một kho mô hình để tự động phiên bản và lưu trữ tất cả các sản phẩm cần thiết để tái tạo mô hình. Các kho mô hình chín chặt như AWS SageMaker và Databrick's MLFlow.

**Giai đoạn 3: Huấn luyện tự động có trạng thái theo lịch trình cố định**

Để đạt được điều này, bạn cần cấu hình lại đoạn mã của mình và một cách để theo dõi dữ liệu và dòng dữ liệu mô hình của bạn. Một ví dụ về phiên bản lịch sử mô hình đơn giản:

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

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

V1.2.12 so với V2.3.43 có nghĩa là đã có 12 lần huấn luyện có trạng thái được thực hiện trên V1.2 và 43 lần trên V2.3.

Bạn có thể cần sử dụng điều này cùng với các kỹ thuật phiên bản khác như phiên bản dữ liệu để theo dõi một bức tranh đầy đủ về cách các mô hình đang phát triển.

Theo tác giả, cô không biết có bất kỳ kho mô hình nào có khả năng theo dõi lịch sử mô hình như vậy nên các công ty thường xây dựng chúng trong nhà.

Tại bất kỳ thời điểm nào, bạn sẽ có nhiều mô hình chạy trong sản xuất cùng một lúc thông qua các sắp xếp được mô tả trong "Kiểm thử mô hình trong Sản xuất".

**Giai đoạn 4: Học liên tục**

Ở giai đoạn này, phần lịch trình cố định của các giai đoạn trước được thay thế bằng cơ chế kích hoạt huấn luyện lại nào đó. Các kích hoạt có thể là:

Dựa trên thời gian

Dựa trên hiệu suất: ví dụ như hiệu suất đã giảm dưới x%

Không phải lúc nào bạn cũng có thể đo trực tiếp độ chính xác trong sản xuất, vì vậy bạn có thể cần sử dụng một số chỉ số proxy yếu hơn.

Dựa trên khối lượng: Tổng số dữ liệu có nhãn tăng thêm 5%

Dựa trên sự chuyển động: ví dụ như khi phát hiện sự thay đổi "lớn" trong phân phối dữ liệu.

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

**2.2. Test production**

Để kiểm thử đầy đủ mô hình của bạn trước khi đưa chúng ra sử dụng rộng rãi, bạn cần cả đánh giá ngoại tuyến trước triển khai VÀ kiểm thử trong môi trường sản xuất. Chỉ có việc đánh giá ngoại tuyến mà không kết hợp với kiểm thử trong môi trường sản xuất là chưa đủ.

Lý tưởng nhất, mỗi nhóm nên xây dựng một quy trình rõ ràng về cách mô hình được đánh giá: các bài kiểm tra nào được thực hiện, ai thực hiện chúng và các ngưỡng áp dụng để thúc đẩy một mô hình lên bước tiếp theo. Điều tốt nhất là nếu những dòng đánh giá này được tự động hóa và kích hoạt khi có cập nhật mô hình mới. Việc thăng cấp từng giai đoạn nên được xem xét tương tự như cách CI/CD được đánh giá trong kỹ thuật phần mềm.

Hai phương pháp phổ biến nhất là

1. Sử dụng một phần chia kiểm thử để so sánh với một mô hình cơ sở và
2. Chạy các kiểm thử ngược.

Phần chia kiểm thử thường là cố định để bạn có một chỉ số so sánh đáng tin cậy để so sánh giữa nhiều mô hình. Điều này cũng có nghĩa là hiệu suất tốt trên một phần chia kiểm thử cố định không đảm bảo hiệu suất tốt trong điều kiện phân phối dữ liệu hiện tại trong môi trường sản xuất.

Kiểm thử ngược là ý tưởng sử dụng dữ liệu được gán nhãn mới nhất mà mô hình chưa nhìn thấy trong quá trình huấn luyện để kiểm thử hiệu suất (ví dụ: nếu bạn đã sử dụng dữ liệu cho ngày cuối cùng, hãy sử dụng dữ liệu cho giờ cuối cùng để kiểm thử ngược lại).

Chúng tôi chưa giới thiệu về kiểm thử ngược lại trước đó. Đây là lần đầu tiên chúng tôi đề cập đến chúng trong bản tóm tắt này.

Có thể dễ bị quen nói rằng kiểm thử ngược lại là đủ để tránh kiểm thử trong môi trường sản xuất, nhưng đó không phải là trường hợp. Hiệu suất trong môi trường sản xuất là nhiều hơn chỉ hiệu suất liên quan đến nhãn. Bạn vẫn cần quan sát những yếu tố như độ trễ, hành vi người dùng với mô hình và tính chính xác của tích hợp hệ thống để đảm bảo mô hình của bạn an toàn khi triển khai rộng rãi.

* + 1. **Testing in Production trong sản xuất**

**Triển Khai Bóng**

Suy luận: Triển khai mô hình thách thức song song với mô hình vô địch hiện tại. Chuyển mọi yêu cầu đến cả hai mô hình nhưng chỉ phục vụ suy luận từ mô hình vô địch. Ghi lại dự đoán của cả hai mô hình để sau đó so sánh chúng.

**Ưu điểm**

* Đây là cách triển khai mô hình an toàn nhất. Ngay cả khi mô hình mới của bạn có lỗi, dự đoán sẽ không được phục vụ.
* Nó có tính đơn giản về mặt khái niệm.
* Cuộc thử nghiệm của bạn sẽ thu thập đủ dữ liệu để đạt được ý nghĩa thống kê nhanh hơn so với tất cả các chiến lược khác vì tất cả các mô hình đều nhận toàn bộ lưu lượng.

**Nhược điểm**

* Phương pháp này không thể được sử dụng khi đo lường hiệu suất của mô hình phụ thuộc vào việc quan sát cách người dùng tương tác với các dự đoán. Ví dụ, các dự đoán từ các mô hình gợi ý bóng sẽ không được phục vụ, vì vậy bạn sẽ không biết liệu người dùng có nhấp vào chúng hay không.
* Phương pháp này tốn kém vì nó làm tăng gấp đôi số lượng dự đoán và do đó làm tăng số lượng tính toán cần thiết.

Nếu suy luận diễn ra bằng bất kỳ chế độ dự đoán trực tuyến nào, bạn sẽ cần tìm cách xử lý các trường hợp như:

Điều gì xảy ra nếu mô hình bóng mất nhiều thời gian hơn mô hình chính để phục vụ một dự đoán?

Nếu mô hình bóng gặp lỗi, liệu mô hình chính cũng nên gặp lỗi hay không?

**A/B Testing**

Triển khai mô hình thách thức cùng với mô hình vô địch (mô hình A) và định tuyến một phần trăm lưu lượng\* đến mô hình thách thức (mô hình B). Dự đoán từ mô hình thách thức được hiển thị cho người dùng. Sử dụng theo dõi và phân tích dự đoán trên cả hai mô hình để xác định xem hiệu suất của mô hình thách thức có độ chệch thống kê so với mô hình vô địch hay không.

Một số trường hợp sử dụng không tương thích với ý tưởng chia lưu lượng và sử dụng nhiều mô hình cùng một lúc. Trong những trường hợp này, kiểm thử A/B có thể được thực hiện bằng cách thực hiện phân chia thời gian: một ngày cho mô hình A, ngày tiếp theo cho mô hình B.

Phân chia lưu lượng phải là một thử nghiệm thực sự ngẫu nhiên. Nếu bạn tích hợp bất kỳ độ chệch lựa chọn nào về việc ai sẽ nhận mô hình A so với mô hình B (như người dùng desktop nhận A và di động nhận B), kết luận của bạn sẽ không chính xác.

Thử nghiệm phải chạy đủ lâu để thu thập đủ mẫu để đạt được độ tin cậy thống kê đủ.

Tính chất thống kê không phải là chứng cứ tuyệt đối (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 bất kỳ mô hình nào.

Không có gì cản trở bạn chạy thử nghiệm A/B/C/D nếu bạn muốn.

**Ưu điểm:**

* Vì dự đoán được phục vụ cho người dùng, phương pháp này cho phép bạn hoàn toàn nắm bắt cách người dùng phản ứng với các mô hình khác nhau.
* Kiểm thử A/B dễ hiểu và có rất nhiều thư viện và tài liệu xung quanh nó.
* Nó rẻ tiền vì có chỉ một dự đoán cho mỗi yêu cầu.
* Bạn không cần phải xem xét các trường hợp biên xuất phát từ việc đồng thời hóa yêu cầu suy luận cho các chế độ dự đoán trực tuyến (xem nhược điểm của triển khai bóng).

**Nhược điểm:**

* Nó không an toàn như Triển khai Bóng. Bạn muốn một bảo đảm đánh giá ngoại tuyến mạnh mẽ hơn rằng mô hình của bạn sẽ không thất bại thảm hại vì bạn sẽ đưa lưu lượng thực vào đó.
* Bạn phải lựa chọn giữa giả sử nhiều rủi ro hơn (định tuyến nhiều lưu lượng đến mô hình B) và thu được đủ mẫu để phân tích nhanh hơn.

**Canary Release**

Triển khai mô hình thách thức và mô hình vô địch cùng một lúc nhưng bắt đầu với mô hình thách thức không nhận lưu lượng. Dần dần chuyển lưu lượng từ mô hình vô địch sang mô hình thách thức (còn được gọi là "canary"). Theo dõi các chỉ số hiệu suất của mô hình thách thức, nếu chúng trông tốt, tiếp tục cho đến khi toàn bộ lưu lượng chuyển sang mô hình thách thức.

Các phiên bản canary có thể được kết hợp với kiểm thử A/B để đo lường chặt chẽ sự khác biệt về hiệu suất.

Các phiên bản canary cũng có thể chạy ở chế độ "YOLO", trong đó bạn đánh giá sự khác biệt về hiệu suất.

Phiên bản khác của canary có thể là việc phát hành mô hình thách thức vào thị trường nhỏ trước và sau đó thực hiện quảng bá cho tất cả các thị trường nếu mọi thứ trông tốt.

Nếu mô hình thách thức bắt đầu gặp vấn đề, định tuyến lại lưu lượng về mô hình vô địch.

**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 đặc điểm chức năng trong công ty của bạn.
* Vì dự đoán từ mô hình thách thức sẽ được phục vụ, bạn có thể sử dụng điều này với các mô hình yêu cầu tương tác của người dùng để nắm bắt hiệu suất.
* So với triển khai bóng, nó rẻ hơn để chạy. Một suy luận cho mỗi yêu cầu.
* Nếu kết hợp với kiểm thử A/B, nó cho phép bạn thay đổi động lực lượng mà mỗi mô hình đang nhận.

**Nhược điểm:**

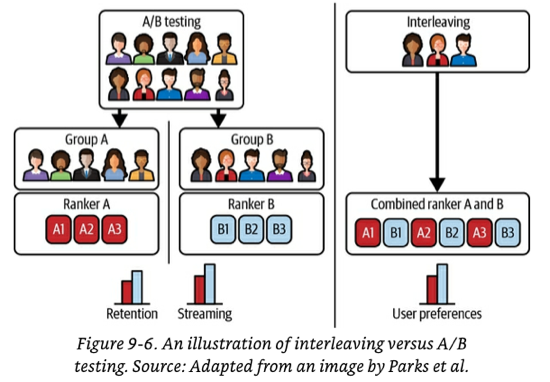
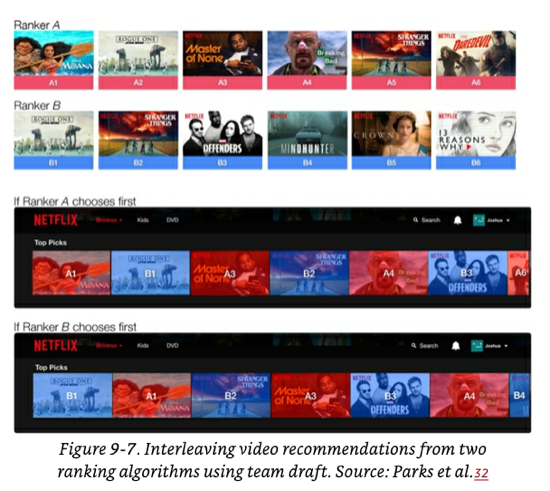
* Nó mở khả năng không chặt chẽ trong việc xác định sự khác biệt về hiệu suất.
* Nếu các phiên bản không được giám sát cẩn thận, tai nạn có thể xảy ra. Có thể coi đây là lựa chọn ít an toàn nhất nhưng rất dễ quay lại trạng thái trước đó.

**Interleaving Experiments**

Trong kiểm thử A/B, một người dùng chỉ nhận dự đoán từ mô hình A hoặc mô hình B. Trong interleaving, một người dùng nhận dự đoán xen kẽ từ cả hai mô hình A và mô hình B. Sau đó, chúng ta theo dõi hiệu suất của mỗi mô hình bằng cách đo lường sự ưa thích của người dùng với dự đoán của từng mô hình (ví dụ: người dùng nhấp nhiều hơn vào các đề xuất từ mô hình B).

Nhiệm vụ đề xuất là một trường hợp sử dụng điển hình cho interleaving. Không phải tất cả các nhiệm vụ đều phù hợp với chiến lược này.

Bạn muốn tránh việc mô hình có ưu thế ưa thích của người dùng không công bằng như luôn chọn lựa dự đoán hàng đầu từ mô hình A. Nó nên có khả năng tương đồng nhau giữa việc khe đầu tiên xuất phát từ mô hình A cũng như từ mô hình B. Các vị trí còn lại có thể được điền bằng phương pháp lựa chọn đồng đội (xem hình).

**Ưu điểm:**

* Netflix đã phát hiện thông qua thử nghiệm rằng interleaving đáng tin cậy xác định mô hình tốt nhất với kích thước mẫu đáng kể nhỏ hơn so với kiểm thử A/B truyền thống.
* Điều này chủ yếu là do cả hai mô hình đều nhận toàn bộ lưu lượng.
* Ngược lại với triển khai bóng, chiến lược này cho phép bạn nắm bắt cách người dùng đang ứng xử với dự đoán của bạn (vì dự đoán được phục vụ).

**Nhược điểm:**

* Triển khai phức tạp hơn so với kiểm thử A/B.
* Bạn cần lo lắng về các trường hợp biên về việc làm gì nếu một trong những mô hình xen kẽ mất quá nhiều thời gian để phản hồi hoặc gặp lỗi.
* Nó làm tăng gấp đôi công suất tính toán cần thiết vì mọi yêu cầu đều nhận 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/gợi ý nhưng không hợp lý cho các nhiệm vụ hồi quy.
* Nó không dễ dàng mở rộng cho một số lượng lớn mô hình thách thức. 2-3 mô hình xen kẽ dường như là điểm lợi thế tốt nhất.

**Bandits**

Bandits là một thuật toán theo dõi hiệu suất hiện tại của mỗi biến thể mô hình và đưa ra quyết định động trên mỗi yêu cầu về việc sử dụng mô hình hiệu suất tốt nhất cho đến nay (tức là tận dụng kiến thức hiện tại) hoặc thử nghiệm bất kỳ mô hình nào khác để có thêm thông tin về chúng (tức là khám phá trong trường hợp một trong những mô hình khác có thể tốt hơn).

Bandits thêm một khái niệm khác vào quyết định về việc sử dụng mô hình nào: chi phí cơ hội.

Bandits không thể áp dụng cho tất cả các trường hợp. Để có thể áp dụng bandits, trường hợp sử dụng của bạn cần:

Thực hiện dự đoán trực tuyến. Dự đoán ngoại tuyến theo lô không tương thích với bandits.

Cần có chuỗi phản hồi ngắn để xác định xem dự đoán có tốt không và cần có cơ chế để truyền phản hồi đó vào thuật toán bandit để cập nhật giá trị của mỗi mô hình.

Có nhiều thuật toán bandit. Đơn giản nhất là gọi là epsilon-greedy. Hai thuật toán mạnh mẽ và phổ biến nhất là Thompson Sampling và Upper Confidence Bound (UCB).

Ưu điểm:

Bandits cần ít dữ liệu hơn so với kiểm thử A/B để xác định mô hình nào tốt hơn. Một ví dụ được đưa ra trong sách là cần 630 nghìn mẫu để đạt được độ tin cậy 95% với kiểm thử A/B so với 12 nghìn với bandits.

Bandits hiệu quả hơn về dữ liệu đồng thời giảm thiểu chi phí cơ hội của bạn. Trong nhiều trường hợp, bandits được coi là tối ưu.

So với kiểm thử A/B, bandits an toàn hơn vì nếu một mô hình thực sự tệ, thuật toán sẽ chọn nó ít hơn. Hơn nữa, sự hội tụ sẽ nhanh chóng, giúp bạn loại bỏ thách thức kém chất lượng nhanh chóng.

Nhược điểm:

So với tất cả các chiến lược khác, bandits khó triển khai hơn nhiều vì cần phải liên tục truyền phản hồi vào thuật toán.

Bandits chỉ có thể sử dụng trong một số trường hợp sử dụng cụ thể (xem phía trên).

Chúng không an toàn như Triển khai Bóng vì những thách thức nhận lưu lượng trực tiếp.