Machine Learning

Project Final

52100916 - Trịnh Lâm Như

** Prepared for, 24/12/2023 **

Abstract

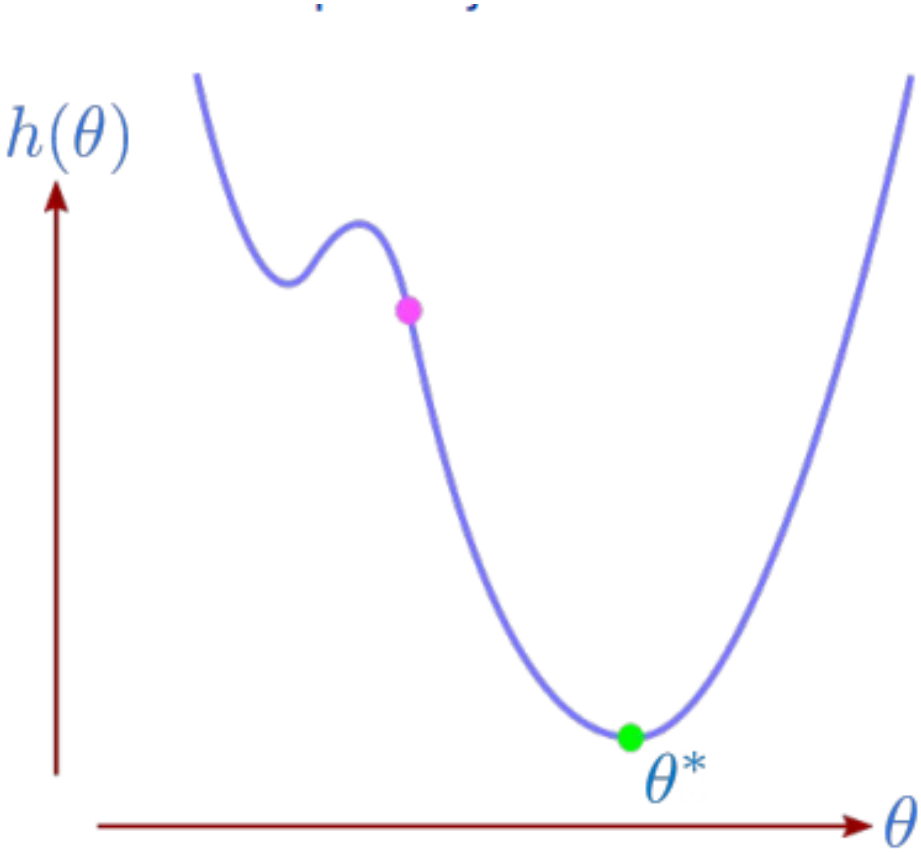
Contents

1 Optimizer

1.1 Thuật toán tối ưu Optimizer

- Trong học máy, thuật toán tối ưu Optimizer là thuật toán hoặc phương pháp cho phép các mô hình học hỏi từ dữ liệu bằng cách điều chỉnh các tham số của chúng trong quá trình huấn luyện.
- Chúng giải quyết vấn đề tối thiểu hóa các hàm đo lường những sai lầm mà mô hình mắc phải như hàm mất mát (lost function) hoặc hàm chi phí (cost function).
- Chúng hoạt động bằng cách thực hiện một chuỗi các thay đổi nhỏ tăng dần đối với các tham số của mô hình, mỗi cái đều được đảm bảo giảm mục tiêu xuống một lượng nhỏ.

- Ví dụ về hàm mục tiêu, trong đó:



  - Trục y: giá trị của hàm mục tiêu $h(\theta)$
  - Trục x: giá trị của tham số θ
  - Mục tiêu của thuật toán tối ưu hóa là giảm thiểu hàm có dạng như này bằng cách di chuyển điểm màu hồng tới khi đạt được mức tối ưu nhất có thể là (điểm màu xanh).

1.2 Gradient Descent

Gradient Descent có thể được coi là một trong những thuật toán phổ biến nhất để tối ưu hóa cho đến nay và cũng phổ biến nhất để tối ưu hóa mạng lưới thần kinh. Thuật toán tối ưu hóa này sử dụng phép tính để sửa đổi các giá trị một cách nhất quán và đạt được mức tối thiểu cục bộ.

Nói một cách đơn giản, hãy coi như bạn đang cầm một quả bóng nằm trên đỉnh một cái bát. Khi bạn đánh mất quả bóng, nó sẽ đi theo hướng dốc nhất và cuối cùng lắng xuống đáy bát. Một gradient cung cấp bóng theo hướng dốc nhất để đạt đến mức tối thiểu cục bộ là đáy bát.

$$xnew = x - \alpha * f'(x)$$

Ở đây α biểu thị khoảng cách di chuyển so với từng gradient sau mỗi lần lặp. 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.

1.2.1 Batch Gradient Descent

Trong Batch Gradient Descent, chúng ta sử dụng toàn bộ tập dữ liệu để tính toán độ dốc của cost function cho mỗi lần lặp lại quá trình giảm gradient và sau đó cập nhật các trọng số.

Vì nếu chúng ta sử dụng toàn bộ tập dữ liệu để tính toán độ hội tụ gradient thì nó sẽ chậm.

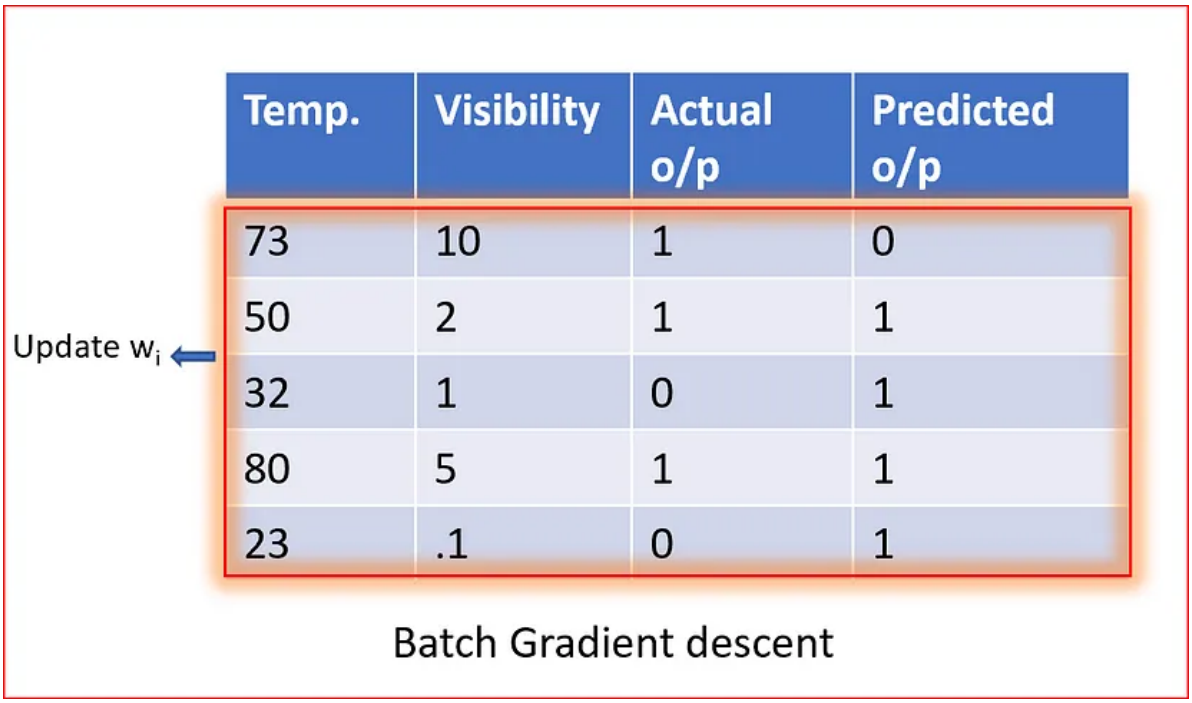


Figure 1: Batch gradient descent uses the entire dataset to calculate each iteration of gradient descent

$$\theta = \theta - \eta.\nabla\theta J(\theta)$$

Vì chúng ta cần tính toán độ dốc cho toàn bộ tập dữ liệu để chỉ thực hiện một lần cập nhật, batch gradient descent có thể rất chậm và khó xử lý đối với các tập dữ liệu không vừa với bộ nhớ.

Nguyên tắc hoạt động:

- BGD tính toán đạo hàm của hàm loss function đối với tất cả các tham số mô hình.

- BGD sử dụng đạo hàm này để cập nhật tất cả các tham số một cách đồng thời theo hướng giảm thiểu hàm loss function.
- BGD lặp lại các bước 1 và 2 cho đến khi đạt được giá trị tối ưu (hoặc khi đạt đến một số lượng iter-lặp nhất định).

1.2.2 Stochastic Gradient Descent

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.

$$\theta = \theta - \eta.\nabla\theta J(\theta; x^i; y^i)$$

Batch Gradient Descent thực hiện các tính toán dư cho các tập dữ liệu lớn, vì nó tính toán lại độ dốc để biết các ví dụ tương tự trước mỗi lần cập nhật tham số. SGD loại bỏ sự dư thừa này bằng cách thực hiện một cập nhật tại một thời điểm. Do đó, nó thường nhanh hơn nhiều. SGD thực hiện cập nhật thường xuyên với phương sai cao khiến hàm mục tiêu dao động mạnh như hình bên dưới.

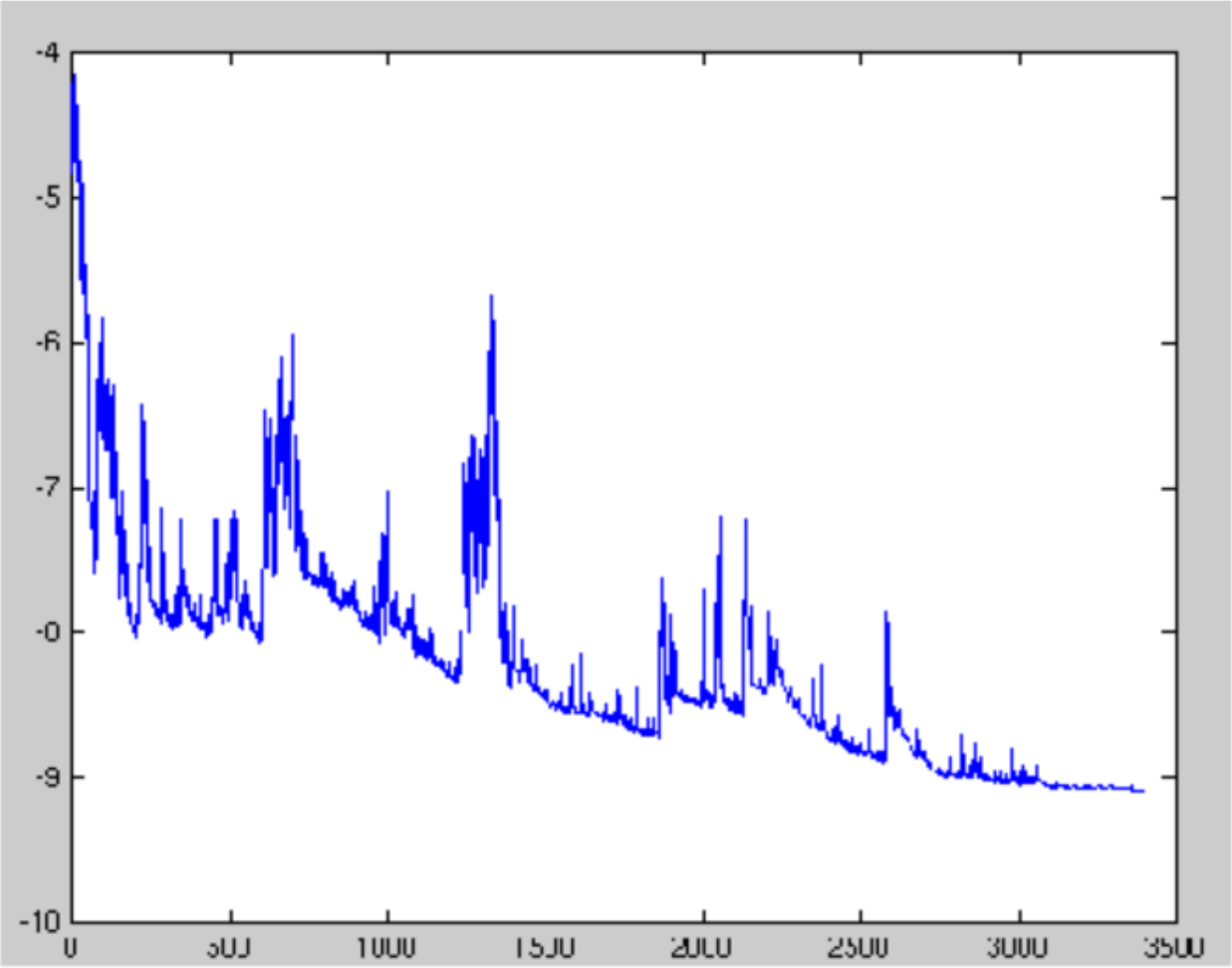


Figure 2: SGD fluctuation (Source: Wikipedia)

Trong khi batch gradient descent hội tụ đến mức tối thiểu của lưu vực, các tham số được đặt vào. Sự biến động của SGD cho phép nó nhảy lên mức tối thiểu cục bộ mới và có khả năng tốt hơn. Mặt khác, điều này cuối cùng sẽ làm phức tạp thêm sự hội tụ đến mức tối thiểu chính xác, vì SGD sẽ giữ vượt mức. Tuy nhiên, người ta đã chứng minh rằng khi chúng ta giảm dần tốc độ học, SGD cho thấy hành vi hội tụ tương tự như việc giảm độ dốc hàng loạt, gần như chắc chắn hội tụ thành một mức

tối thiểu cục bộ hoặc toàn cầu để tối ưu hóa không lồi và lồi tương ứng.

1.2.3 Mini-Batch Gradient Descent

Khác với SGD, mini-batch sử dụng mọt số lượng n lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu N rất nhiều). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia dữ liệu thành các mini-batch, mỗi mini-batch có n điểm dữ liệu (trừ mini-batch cuối cùng có thể ít hơn nếu N không chia hết cho n). Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật. Công thức có thể viết dưới dạng:

$$\theta = \theta - \eta.\nabla\theta J(\theta; x^{i:i+n}; y^{i:i+n})$$

Với $x^{i:i+n}$ được hiểu là dữ liệu từ thứ i đến thứ i + n − 1 (theo ký hiệu của Python). Dữ liệu này sau mỗi epoch khác nhau chúng ta cần được xáo trộn. Mini-Batch GD được sử dụng trong hầu hết các thuật toán của Machine Learning, đặc biệt là Deep Learing. Giá trị n thường được lựa chọn khoảng 50 đến 256. Nhìn chung, gradient descent làm giảm phương sai cập nhật tham số (hội tụ ổn định hơn) so với SGD. Mini-Batch GD thường là thuật toán được lựa chọn khi huấn luyện mạng lưới thần kinh và thuật ngữ SGD cũng thường được sử dụng khi sử dụng mini-batches.

1.3 Momentum methods

Ý tưởng cơ bản của Momentum trong ML là tăng tốc độ huấn luyện. Khái niệm này là một trong những chuông và còi nhỏ mà bạn cho là không quan trọng nhưng hóa ra lại tiết kiệm thời gian thực sự và khiến mọi việc diễn ra suôn sẻ hơn rất nhiều. Nó

chủ yếu được sử dụng trong các mạng thần kinh vì kích thước của dữ liệu trong NN tạo ra sự khác biệt đáng kể về thời gian trong khi huấn luyện độ dốc. Momentum là một phương pháp giúp tăng tốc SGD theo hướng thích hợp và làm giảm dao động như có thể thấy trong hình bên dưới. Nó thực hiện điều này bằng cách thêm một phần

$$\gamma$$

của vectơ cập nhật của bước thời gian qua đến vectơ cập nhật hiện tại.

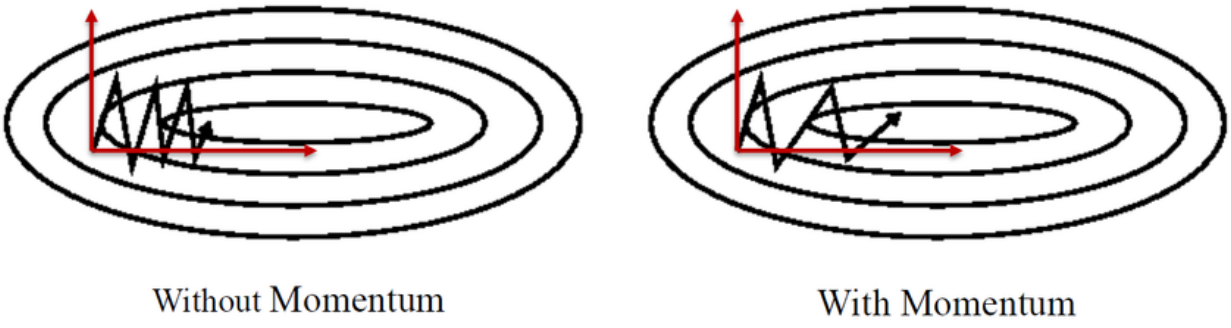


Figure 3: Momentum

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

γ thường được đặt bằng 0,9 hoặc một giá trị tương tự.

Về cơ bản, khi sử dụng động lượng, chúng ta đẩy một quả bóng xuống một ngọn đồi. Quả bóng tích lũy động lượng khi nó lăn xuống dốc, trên đường đi càng lúc càng nhanh hơn (cho đến khi đạt vận tốc cuối, nếu có lực cản của không khí, tức là $\gamma < 1$). Điều tương tự cũng xảy ra với các cập nhật tham số của chúng ta: Động lượng tăng đối với các kích thước có độ dốc hướng theo cùng một hướng và giảm cập nhật

cho kích thước có độ dốc thay đổi hướng. Kết quả là chúng ta đạt được sự hội tụ nhanh hơn và giảm dao động.

1.4 RMSprop

RMSprop là một phương pháp stochastic mini-batch phiên bản đổi mới và tiến bộ hơn. RMSprop (Root Mean Squared Propagation) giúp cải thiện tốc độ hội tụ và tính ổn định của quá trình huấn luyện mô hình.

Cũng như các thuật toán Gradient descent khác, RMSprop hoạt động bằng cách tính toán độ dốc của hàm mất mát (lost function) theo các tham số của mô hình và cập nhật các tham số theo hướng ngược lại với độ dốc để giảm thiểu tổn thất. Tuy nhiên, RMSProp giới thiệu một số kỹ thuật bổ sung để cải thiện hiệu suất của quá trình tối ưu hóa.

Đặc điểm chính là việc sử dụng đường trung bình động của gradient bình phương để chia tỷ lệ học tập cho từng tham số. Điều này giúp ổn định quá trình học và ngăn chặn sự dao động trong quỹ đạo tối ưu hóa.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \tag{1}$$

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

Trong đó:

- $E[g^2]_t$ là đường trung bình động của gradient bình phương;

- 0.9 là siêu tham số kiểm soát tốc độ phân rã của đường trung bình động;
- η là siêu tham số kiểm soát kích thước bước cập nhật;
- g_t là gradient của hàm mất mát đối với tham số;
- ϵ là một hằng số nhỏ được thêm vào mẫu số để ngăn việc chia cho 0.

1.5 Adam

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.

Một trong những thành phần chính của Adam là các trung bình động trọng số mũ (hay còn được gọi là trung bình rò rỉ) để ước lượng cả động lượng và mô-men bậc hai của gradient. Cụ thể, nó sử dụng các biến trạng thái

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4}$$

Ở đây β_1 và β_2 là các tham số trong số không âm. Các lựa chọn phổ biến cho chúng là $\beta_1 = 0.9$ và $\beta_2 = 0.999$. Điều này có nghĩa là ước lượng phương sai di chuyển chậm hơn nhiều so với số hàng đồng lượng. Lưu ý rằng nếu ta khởi tạo $m_0 = v_0 = 0$, thuật toán sẽ có độ chệch ban đầu đáng kể về các giá trị nhỏ hơn. Vấn đề này có thể được

giải quyết bằng cách sử dụng $\sum_{i=0}^{t} \beta^i = \frac{1-\beta^t}{1-\beta}$ để chuẩn hóa lại các số hàng. Tương tự, các biến trạng thái được chuẩn hóa như sau

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{và} \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Sau đó, cập nhật các tham số giống như chúng ta đã làm với RMSprop, mang lại quy tắc cập nhật Adam như sau:

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

Giá trị mặc định là 0,9 đối với β_1, 0,999 cho β_2 và 10^{-8} cho ϵ. Chúng cho thấy rằng Adam hoạt động tốt trong thực tế và có lợi thế hơn so với các thuật toán phương pháp học tập thích ứng khác.

1.6 So sánh Batch, Stochastic và Mini-batch gradient descent

- Độ hội tụ (Convergence)

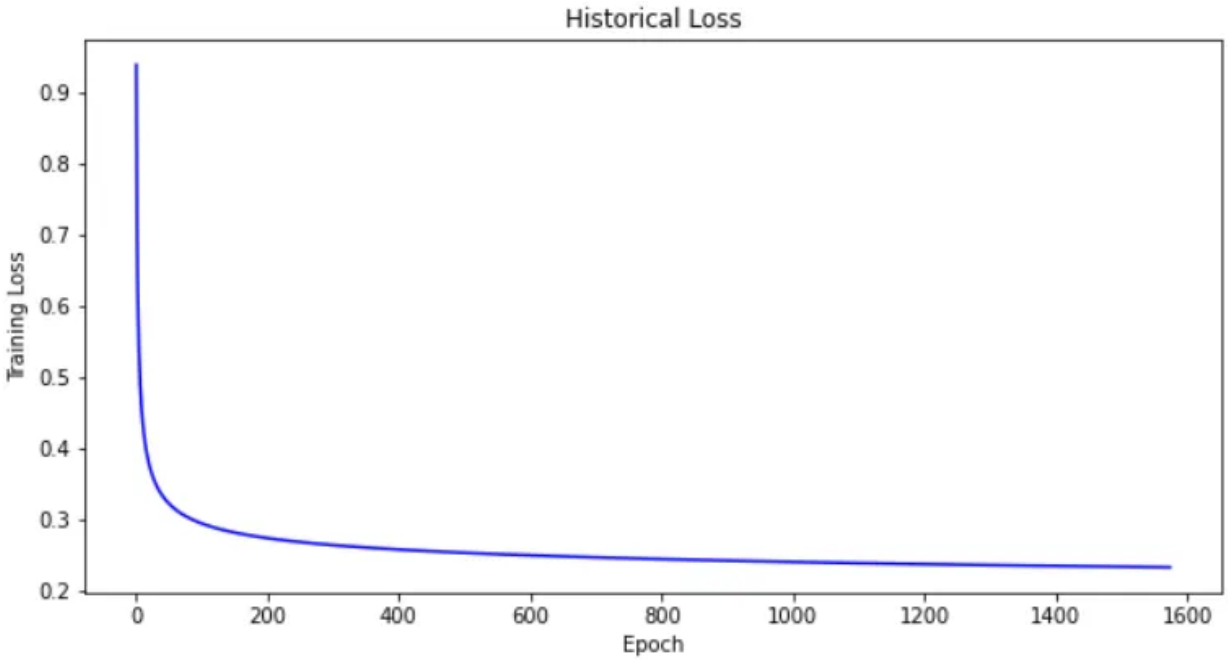


Figure 4: Batch gradient descent: Loss versus epoch

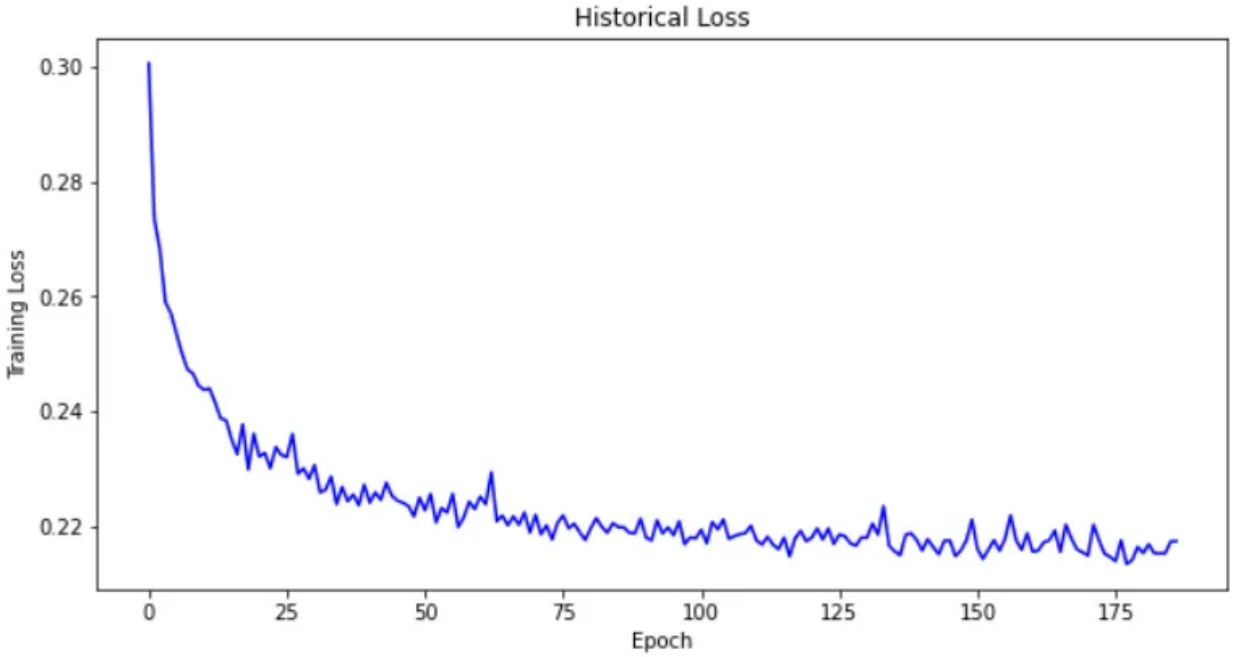


Figure 5: Stochastic gradient descent: Loss versus epoch

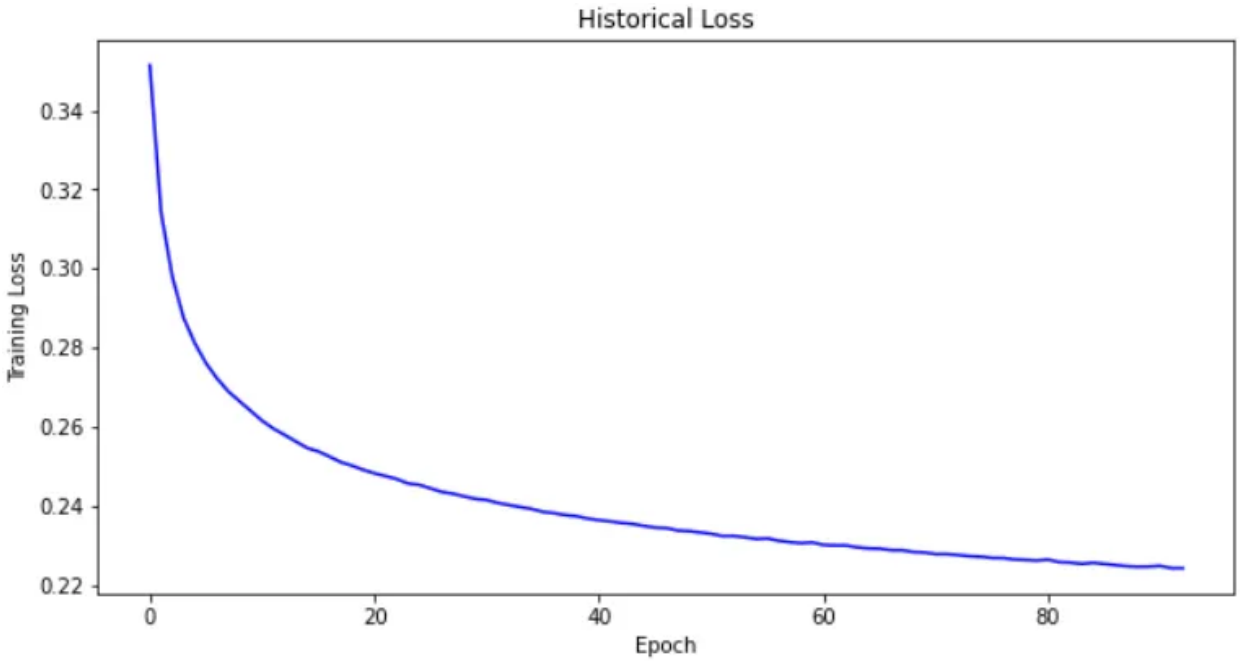


Figure 6: Mini-batch gradient descent: Loss versus epoch

Ta có thể thấy, batch gradient descent có đường hội tụ ổn định trong khi stochastic gradient descent có độ nhiễu khá mạnh và mini-batch grandient descent có độ nhiễu tương đối không đáng kể.

Thời gian hội tụ trong quá trình huấn luyện là:

- Batch gradient descent: 727.65 seconds
- Stochastic gradient descent: 544.12 seconds
- Mini-batch gradient descent: 45.66 seconds

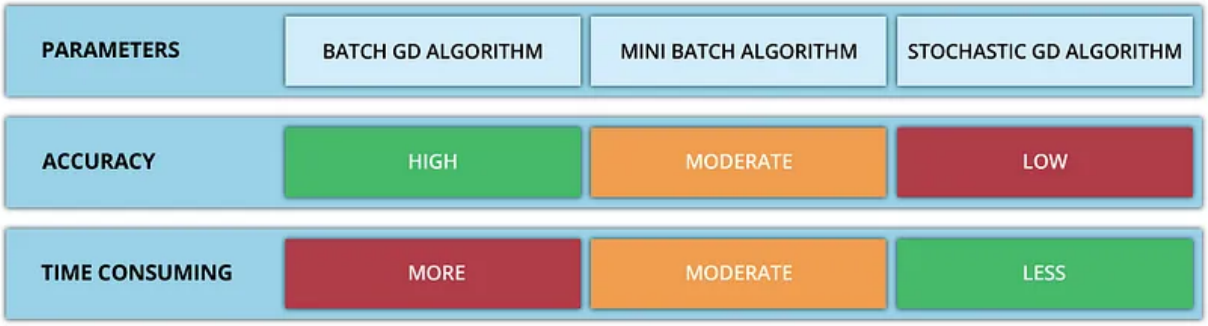


Figure 7: The accuracy scores

Đối với stochastic gradient descent, có mỗi điểm dữ liệu một lần cập nhật và ví dụ trong trường hợp, có 60000 dữ liệu huấn luyện, thì sẽ có 60000 bản cập nhật mỗi lần lặp. Do đó, phương pháp stochastic gradient descent có thể đạt được sự hội tụ tương đối nhanh hơn với độ nhiễu lớn hơn so với phương pháp batch gradient descent.

Còn đối với mini-batch gradient descent, mỗi b điểm dữ liệu là một lần cập nhật. Giả sử chọn b=64, do đó sẽ có 938 cập nhật mỗi lần lặp. Bằng cách xem xét nhiều dữ liệu hơn cho mỗi lần cập nhật trong mỗi lần lặp, nhiễu được tạo ra sẽ ít hơn so với stochastic gradient descent. Với ít nhiễu hơn so với phương pháp stochastic gradient descentn và nhiều cập nhật hơn trên mỗi lần lặp so với phương pháp batch gradient descent, mini-batch gradient descent có thể đạt được độ hội tụ nhanh nhất đáng kể.

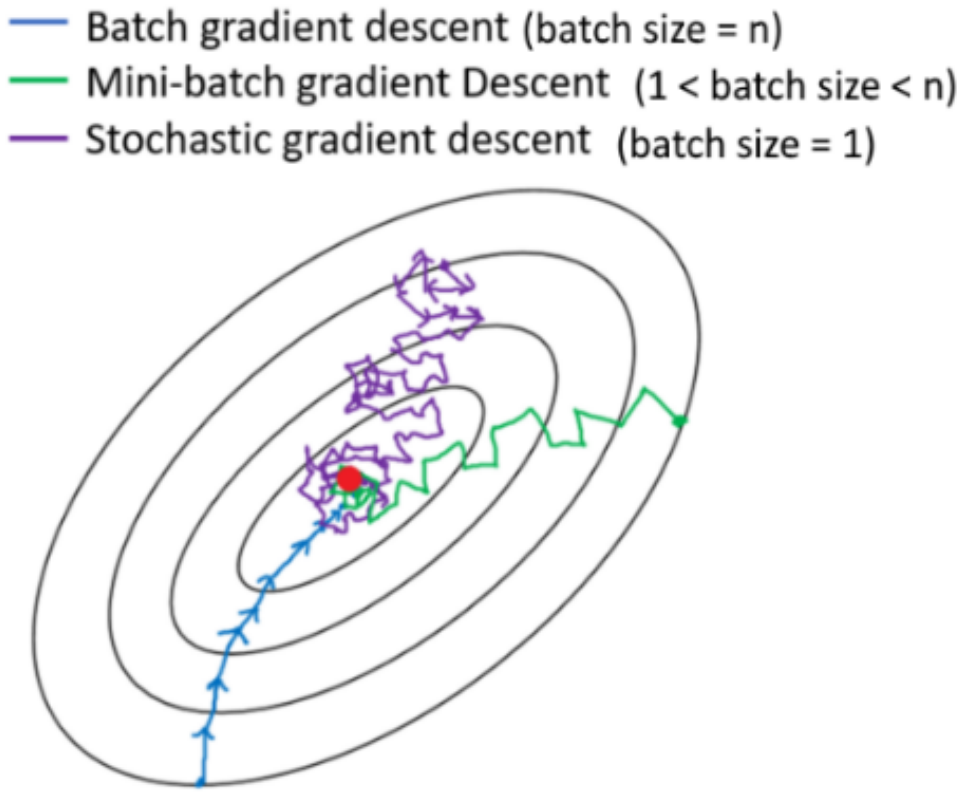


Figure 8: Trajectory of Batch , Mini Batch and Stochastic gradient descent

- Số lượng dữ liệu cho mỗi lần update

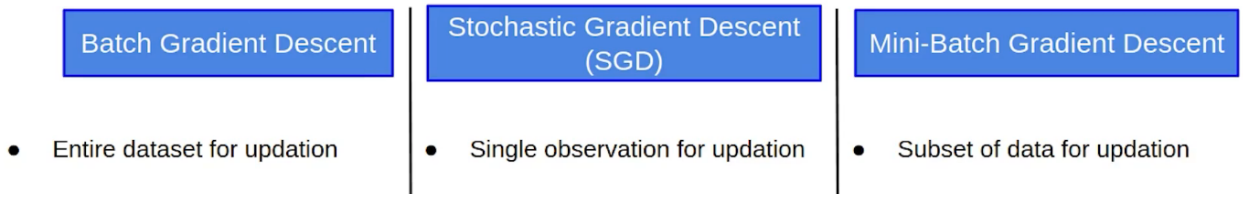


Figure 9: Number of observations used for Updation

  - Trong batch gradient Descent, chúng ta lấy toàn bộ tập dữ liệu > tính hàm chi phí > cập nhật tham số.

– Trong trường hợp Stochastic Gradient Descent, ta cập nhật các tham số sau mỗi lần quan sát và biết rằng khi các trọng số được cập nhật, nó được gọi là một lần lặp.

– Trong trường hợp Mini-batch Gradient Descent, ta chỉ lấy một tập hợp con dữ liệu và cập nhật các tham số dựa trên mỗi tập hợp con.

- Cost function

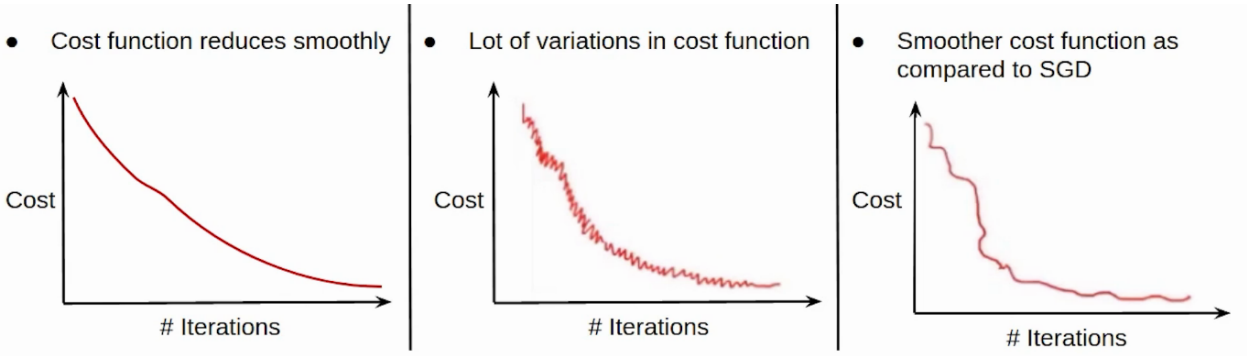


Figure 10: Cost function

– Cập nhật các tham số bằng cách sử dụng toàn bộ tập dữ liệu trong trường hợp Batch GD, nên hàm chi phí, trong trường hợp này, sẽ giảm đi một cách trơn tru.

– Mặt khác, bản cập nhật này trong trường hợp của SGD không được suôn sẻ cho lắm. Vì ta đang cập nhật các tham số dựa trên một quan sát duy nhất nên có rất nhiều lần lặp lại. Cũng có thể mô hình cũng bắt đầu học tiếng ồn.

– Việc cập nhật hàm chi phí trong trường hợp Mini-batch Gradient Descent mượt mà hơn so với hàm chi phí trong SGD. Vì không cập nhật các tham

số sau mỗi lần quan sát mà sau mỗi tập hợp con dữ liệu.

- Chi phí và thời gian tính toán

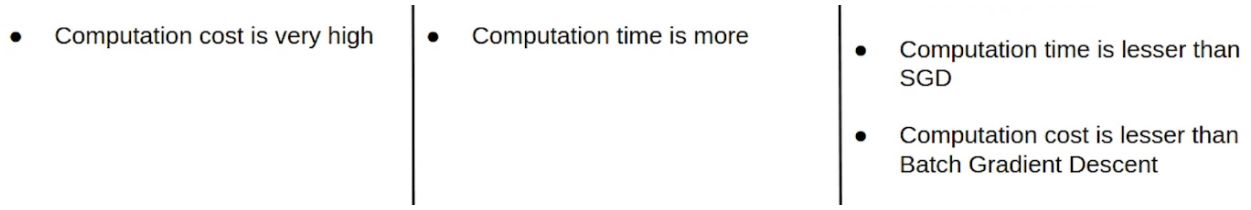


Figure 11: Computation Cost and Time

  - Vì phải tải toàn bộ tập dữ liệu cùng một lúc, thực hiện việc truyền tiến trên đó và tính toán lỗi, sau đó cập nhật các tham số, nên chi phí tính toán trong trường hợp Batch gradient descent là rất cao.
  - Chi phí tính toán trong trường hợp SGD ít hơn so với Batch gradient descent vì chúng tôi phải tải từng quan sát tại một thời điểm nhưng thời gian tính toán ở đây sẽ tăng lên vì sẽ có nhiều cập nhật hơn, dẫn đến số lần lặp lại nhiều hơn .
  - Trong trường hợp Mini-batch Gradient Descent, việc lấy một tập hợp con dữ liệu sẽ có số lần lặp hoặc cập nhật ít hơn và do đó thời gian tính toán trong trường hợp Mini-batch Gradient Descent sẽ nhỏ hơn SGD. Ngoài ra, vì không tải toàn bộ tập dữ liệu cùng một lúc trong khi tải một tập hợp con dữ liệu nên chi phí tính toán cũng thấp hơn so với phương pháp Batch gradient descent.

1.7 Cách chọn phù hợp

- Loại hàm loss function: Một số phương pháp Optimizer phù hợp hơn với các loại hàm loss function nhất định. Ví dụ, SGD thường được sử dụng cho các hàm loss function convex, trong khi Adam thường được sử dụng cho các hàm loss function non-convex.

- Kích thước của dataset: Các phương pháp Optimizer dựa trên GD thường chậm đối với các dataset lớn. Trong trường hợp này, các phương pháp Optimizer dựa trên SGD hoặc MBGD thường được sử dụng.

- Yêu cầu về thời gian huấn luyện: Nếu cần huấn luyện mô hình nhanh chóng, các phương pháp Optimizer dựa trên SGD hoặc MBGD thường được sử dụng. Nếu có đủ thời gian, các phương pháp Optimizer dựa trên GD có thể giúp mô hình đạt được hiệu quả tốt hơn.

2 Continual Learning và Test Production

https://github.com/serodriguez68/designing-ml-systems-summary/blob/main/09-continual-learning-and-test-in-production.md

2.1 Continual Learning

Continual Learning tập trung vào việc cập nhật mô hình khi có dữ liệu mới, điều này làm cho mô hình theo kịp các phân phối dữ liệu hiện tại. Sau khi mô hình được cập nhật, nó không thể được phát hành một cách mù quáng 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 sử dụng.

Continual Learning thường bị hiểu sai:

- Continual Learning 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 "online learning algorithms".
  - Khái niệm Continual Learning có thể được áp dụng cho bất kỳ thuật toán ML được giám sát nào.
- Continual Learning KHÔNG có nghĩa là bắt đầu công việc huấn luyện 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 continual learning đều cập nhật mô hình của họ theo từng đợt nhỏ (ví dụ cứ sau 512 hoặc 1024). Số lượng ví dụ phụ thuộc vào từng tác vụ.

Nhìn bề ngoài, Continual learning có thể giống như một nhiệm vụ của data scientist. Tuy nhiên, nó đòi hỏi rất nhiều công việc về cơ sở hạ tầng để kích hoạt nó.

2.1.1 Thách thức

Continual learning đã được áp dụng trong công nghiệp với thành công lớn. Tuy nhiên, nó có vài thách thức lớn mà các công ty cần phải vượt qua.

- **Truy cập dữ liệu mới**

  Nếu muốn cập nhật mô hình của mình mỗi giờ. Chúng ta cần dữ liệu huấn luyện được dán nhãn chất lượng sau mỗi giờ. Nhịp độ cập nhật càng ngắn thì thách thức này càng trở nên nghiêm trọng.

  – **Vấn đề: Tốc độ lưu trữ dữ liệu vào kho dữ liệu**

    Nhiều công ty lấy dữ liệu huấn luyện 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.

    Ví dụ: các phần dữ liệu trong kho có thể được lấy trực tiếp từ quá trình vận chuyển theo thời gian thực nhưng một phần dữ liệu có thể đến từ các ETL hàng ngày hoặc hàng tuần sao chép dữ liệu từ các nguồn khác.

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 huấn luyện trước khi đưa vào kho lưu trữ. Điều này đặc biệt mạnh mẽ khi việc vận chuyển thời gian thực được nối vào một cửa hàng tính năng. Có một số thách thức để đạt được điều này:

* Có thể không bơm tất cả dữ liệu của mình qua các sự kiện. Điều này đặc biệt phổ biến đối với dữ liệu nằm trong hệ thống của nhà cung cấp bên ngoài mà bạn không thể kiểm soát. Nếu bạn tin rằng dữ liệu đó là mới, bạn sẽ cần tìm ra cách nắm bắt các thay đổi trên các hệ thống đó bằng cách sử dụng sự kiện. Web-hook hoặc API thăm dò ý kiến để phát lại các sự kiện là những giải pháp phổ biến.
* Ở một số công ty, ETL theo đợt có xu hướng thực hiện nhiều công việc xử lý nặng nhọc và kết hợp dữ liệu để đưa dữ liệu vào kho dữ liệu để làm cho dữ liệu trở nên hữu ích hơn. Nếu bạn thay đổi sang chiến lược truyền tải toàn thời gian thực, bạn cần tìm ra cách thực hiện quá trình xử lý tương tự trên một luồng dữ liệu.

– **Vấn đề: Tốc độ ghi nhãn**

* Tốc độ để gắn nhãn dữ liệu mới thường là điểm nghẽn. Những ứng cử viên tốt nhất cho việc học tập liên tục là những nhiệm vụ được gắn nhãn tự nhiên với các vòng phản hồi ngắn (vòng phản hồi càng ngắn thì bạn có thể gắn nhãn càng nhanh).
* Nếu không dễ dàng có được nhãn tự nhiên trong khung thời gian cần thiết, bạn cũng có thể thử các kỹ thuật giám sát yếu hoặc bán giám

sát để có được nhãn kịp thời (với chi phí là nhãn có thể ồn ào hơn). Phương án cuối cùng là bạn có thể xem xét chú thích nhãn sử dụng nguồn lực cộng đồng nhanh chóng và thường xuyên.

* Một yếu tố khác ảnh hưởng đến tốc độ ghi nhãn là chiến lược tính toán nhãn:
  - Bạn có thể thực hiện tính toán nhãn theo đợt. Các công việc hàng loạt này thường chạy định kỳ trên dữ liệu đã được gửi vào kho dữ liệu. Do đó, tốc độ ghi nhãn là một hàm của cả tốc độ lưu trữ dữ liệu và nhịp độ của công việc tính toán nhãn.
  - Tương tự như giải pháp ở trên, một cách tiếp cận phổ biến để tăng tốc độ ghi nhãn là tính toán nhãn trực tiếp từ việc vận chuyển (sự kiện) thời gian thực. Tính toán phát trực tuyến này có những thách thức riêng.

- **Đánh giá**
  - Việc áp dụng việc học tập liên tục như một phương pháp thực hành có nguy cơ dẫn đến những thất bại của mô hình. Việc cập nhật mô hình càng thường xuyên thì càng có nhiều cơ hội để mô hình thất bại.
  - Ngoài ra, việc học hỏi liên tục sẽ mở ra cơ hội cho các cuộc tấn công đối nghịch phối hợp nhằm đầu độc các mô hình. Điều này có nghĩa là việc thử nghiệm các mô hình trước khi triển khai chúng cho nhiều đối tượng hơn là rất quan trọng.
  - Việc kiểm tra cần có thời gian nên đây có thể là một yếu tố hạn chế khác

về tần suất cập nhật mô hình nhanh nhất có thể nhận được.

  - Ví dụ: Một mô hình mới để phát hiện gian lận có thể mất khoảng 2 tuần để có đủ lưu lượng truy cập để được đánh giá một cách đáng tin cậy.

- **Chia tỉ lệ dữ liệu**

  - Tính toán thường yêu cầu chia tỷ lệ. Chia tỷ lệ thường yêu cầu quyền truy cập vào số liệu thống kê dữ liệu toàn cầu như tối thiểu, tối đa, trung bình và phương sai.

  - Nếu sử dụng phương pháp đào tạo trạng thái, số liệu thống kê toàn cầu phải xem xét cả dữ liệu trước đó đã được sử dụng để huấn luyện mô hình cộng với dữ liệu mới đang được sử dụng để làm mới mô hình. Việc theo dõi số liệu thống kê toàn cầu trong trường hợp này có thể khó khăn.

  - Một kỹ thuật phổ biến để thực hiện việc này là tính toán hoặc ước tính các thống kê này tăng dần khi quan sát dữ liệu mới (ngược lại với việc tải toàn bộ tập dữ liệu vào thời gian đào tạo và tính toán từ đó).

- **Thuật toán**

  - Vấn đề này xuất hiện khi sử dụng một số loại thuật toán nhất định và muốn cập nhật chúng thật nhanh (ví dụ sau mỗi giờ).

  - Các thuật toán được đề cập là những thuật toán đòi hỏi quyền truy cập vào tập dữ liệu đầy đủ để được huấn luyện. Ví dụ: các mô hình dựa trên ma trận, dựa trên giảm kích thước và dựa trên cây. Những loại mô hình này không thể được huấn luyện tăng dần bằng dữ liệu mới như mạng thần kinh hoặc các mô hình dựa trên trọng số khác có thể.

– Vấn đề chỉ xảy ra khi ta cần cập nhật chúng thật nhanh vì không thể chờ thuật toán xem hết toàn bộ tập dữ liệu.

– Có một số biến thể của các mô hình bị ảnh hưởng đã được thiết kế để huấn luyện tăng dần, nhưng việc áp dụng các thuật toán này không phổ biến. Một ví dụ là Cây Hoeffding và các biến thể phụ.

2.1.2 Khái niệm Stateless retraining VS Stateful training

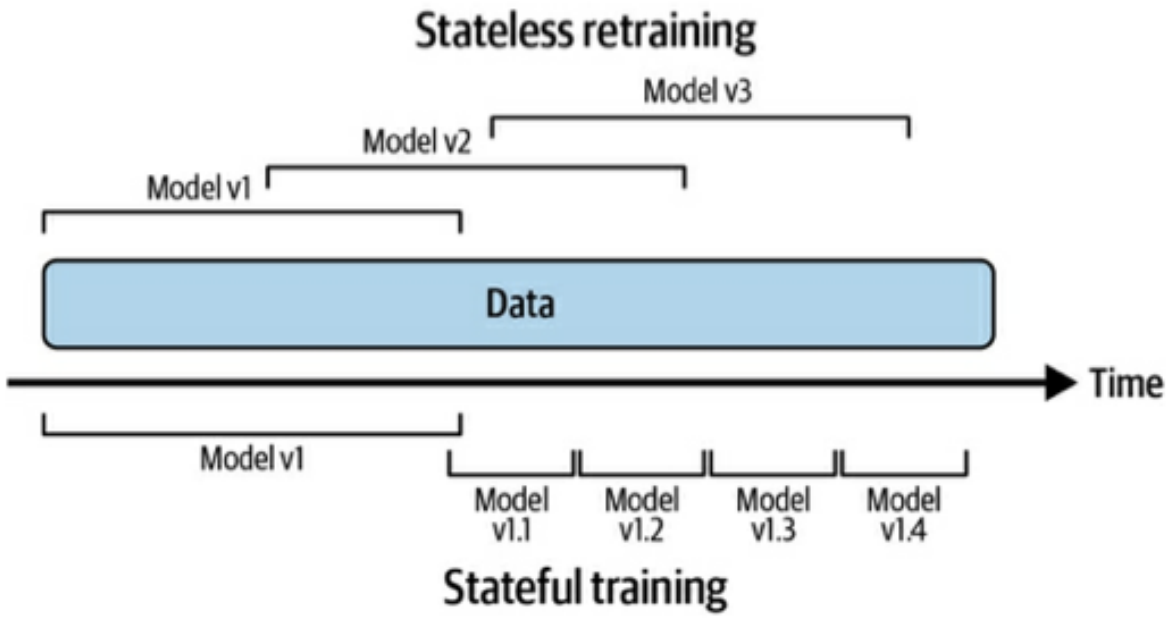


Figure 9-2. Stateless retraining versus stateful training

Figure 12: Stateless retraining VS Stateful training

1. Stateless retraining - Huấn luyện lại không trạng thái

   Huấn luyện lại mô hình 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 huấn luyện lại không trạng thái.

2. Stateful training - Huấn luyện có trạng thái

   Khởi tạo mô hình với các trọng số từ vòng huấn luyện trước và tiếp tục huấn luyện bằng cách sử dụng dữ liệu mới chưa từng thấy.

   - 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 năng lượng tính toán hơn.
   - Về mặt lý thuyết, nó có thể tránh việc lưu trữ dữ liệu hoàn toàn sau khi dữ liệu đã được sử dụng để huấn luyện (và để lại một khoảng thời gian an toàn). Điều này giúp loại bỏ những lo ngại về quyền riêng tư dữ liệu.
   - Thỉnh thoảng cần phải chạy huấn luyện lại không trạng thái với một lượng lớn dữ liệu để hiệu chỉnh lại mô hình.
   - Sau khi cơ sở hạ tầng được thiết lập chính xác, việc thay đổi từ huấn luyện lại không trạng thái sang huấn luyện có trạng thái sẽ trở thành một nút nhấn.
   - **Lặp lại mô hình so với lặp lại dữ liệu:** Huấn luyện trạng thái chủ yếu được sử dụng để kết hợp dữ liệu mới vào kiến trúc mô hình cố định và hiện có (tức là lặp lại dữ liệu). Nếu muốn thay đổi các tính năng hoặc kiến trúc

của mô hình, bạn sẽ cần phải thực hiện quá trình huấn luyện lại không trạng thái lần đầu tiên.

2.1.3 Các giai đoạn chính của Continual Learning

Các công ty thường tiến hành Continual Learning theo bốn giai đoạn.

Giai đoạn 1: Huấn luyện lại một cách thủ công

Các mô hình chỉ được huấn luyện 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) nhóm của bạn có thời gian để cập nhật mô hình.

Giai đoạn 2: Huấn luyện lại trạng thái tự động theo lịch trình không 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 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 "trực giác".
- Điểm uốn giữa giai đoạn 1 và giai đoạn 2 thường là một tập lệnh do ai đó viết để chạy quá trình huấn luyện 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:

    1. Kéo dữ liệu

2. Xuống mẫu hoặc lấy mẫu dữ liệu nếu cần thiết
3. Trích xuất các tính năng
4. Xử lý và/hoặc chú thích nhãn để tạo dữ liệu huấn luyện
5. Bắt đầu quá trình huấn luyện
6. Đánh giá mô hình mới
7. Triển khai sử dụng

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

Để đạt được điều này, ta cần phải cấu hình lại tập lệnh của mình và cách theo dõi dòng dữ liệu cũng như mô hình của mình. 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 huấn luyện 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 huấn luyện trạng thái được thực hiện trên V1.2 và 43 khóa huấn luyện được thực hiện trên V2.3.

Giai đoạn 4: Học 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 huấn luyện 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

- Dựa trên khối lượng: Tổng lượng dữ liệu được dán nhãn tăng 5
- Dựa trên sự trôi dạt: ví dụ khi phát hiện 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.

2.1.4 Tần số cập nhật mô hình

Chúng ta cần hiểu và xác định mức lợi ích ta nhận được khi cập nhật mô hình của mình với dữ liệu mới. Muốn đạt được càng nhiều thì càng phải huấn luyện lại thường xuyên.

Đo lường độ mới của dữ liệu

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.

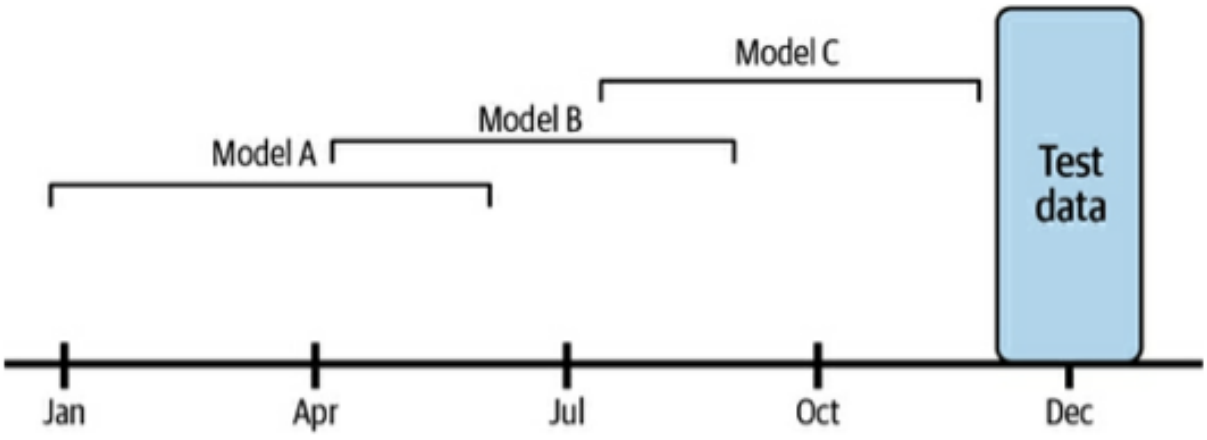


Figure 9-5. To get a sense of the performance gain you can get from fresher data, train your model on data from different time windows in the past and test on data from today to see how the performance changes

Figure 13: Examples with multi-month datasets

Nếu bạn phát hiện ra rằng việc để mô hình cũ trong 3 tháng sẽ gây ra sự khác biệt 10% về độ chính xác của dữ liệu thử nghiệm hiện tại và 10% là không thể chấp nhận được, thì bạn cần huấn luyện lại sau chưa đầy 3 tháng.

Lặp lại mô hình khi nào

- Nếu ta tiếp tục giảm kích hoạt đào tạo lại việc lặp lại dữ liệu và không thu được nhiều lợi ích, có lẽ bạn nên đầu tư vào việc tìm kiếm một mô hình tốt hơn (tất nhiên là nếu doanh nghiệp của bạn cần một mô hình).
- 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.

2.2 Test Production

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

Lý tưởng nhất là mỗi nhóm đưa ra một quy trình rõ ràng về cách đánh giá các mô hình: thử nghiệm nào sẽ chạy, ai thực hiện chúng và các ngưỡng áp dụng để thúc đẩy mô hình lên giai đoạn tiếp theo. Tốt nhất là các quy trình đánh giá này được tự động hóa và khởi động khi có bản cập nhật mô hình mới. Việc thăng cấp giai đoạn cần được xem xét tương tự như cách đánh giá CI/CD trong công nghệ phần mềm.

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.

- **Test splits** 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.

- **Backtesting - Kiểm tra ngược** là ý tưởng sử dụng dữ liệu được gắn nhãn mới nhất mà mô hình chưa thấy trong quá trình huấn luyện để kiểm tra hiệu suất (ví dụ: nếu đã sử dụng dữ liệu của ngày cuối cùng, hãy sử dụng dữ liệu của giờ cuối cùng để kiểm tra lại).

Chiến lược Testing in Production Strategies

1. **Shadow Deployment:** Triển khai mô hình thách đấu song song với mô hình

tốt nhất hiện có. Gửi mọi yêu cầu đến cả hai mô hình nhưng chỉ phục vụ suy luận của mô hình tốt nhất. 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 có lỗi, dự đoán sẽ không được thực hiện.
- Nó đơn giản về mặt khái niệm.
- Thử nghiệm 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 để 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 mô hình đề xuất bóng tối sẽ không được cung cấp nên bạn sẽ không thể biết liệu người dùng có nhấp vào chúng hay không.
- 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. **A/B Testing:** triển khai mô hình thách thức cùng với mô hình quán quân (mô hình A) và định tuyến phần trăm lưu lượng truy cập đến người thách thức (mô hình B). Dự đoán từ người thách đấu 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

thành tích của người thách đấu có tốt hơn về mặt thống kê so với nhà vô địch hay không.

Ưu điểm:

- Vì dự đoán được cung cấp cho người dùng nên kỹ thuật này cho phép nắm bắt đầy đủ cách người dùng phản ứng với các mô hình khác nhau.
- Testing A/B rất dễ hiểu và có rất nhiều thư viện cũng như tài liệu hỗ trợ.
- 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 *(shadow deploy)*).

Nhược điểm:

- Nó kém an toàn hơn so với Shadow Deployment.
- Quyền lựa chọn cố hữu 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.