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

TÌM HIỂU VỀ PHƯƠNG PHÁP OPTIMIZER, CONTINUAL LEARNING VÀ TEST PRODUCTION TRONG MÔ HÌNH HỌC MÁY

Người hướng dẫn: **THẦY LÊ ANH CƯỜNG**
Người thực hiện: **NGUYỄN THỊ DIỄM SƯƠNG – 52000129**
Lớp **: 20050201**
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 GIỮA KỲ NHẬP MÔN HỌC MÁY

TÌM HIỂU VỀ PHƯƠNG PHÁP OPTIMIZER, CONTINUAL LEARNING VÀ TEST PRODUCTION TRONG MÔ HÌNH HỌC MÁY

Người hướng dẫn: **THẦY LÊ ANH CƯỜNG**
Người thực hiện: **NGUYỄN THỊ DIỄM SƯƠNG – 52000129**
Lớp **: 20050201**
Khoá **: 24**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Trong thời gian hoàn thành bài báo cáo vừa qua, em đã nhận được rất nhiều sự giúp đỡ, hướng dẫn và hỗ trợ tận tình từ quý thầy cô và các bạn. Em cảm thấy vô cùng biết ơn và muốn gửi những lời cảm ơn sâu sắc đến thầy cô, bạn bè và gia đình đã giúp bài báo cáo của em đạt kết quả tốt như hiện nay.

Đặc biệt, em muốn bày tỏ sự biết ơn sâu sắc và lòng kính trọng đến thầy Lê Anh Cường, thầy là người đã giảng dạy em môn Nhập môn Học máy trong suốt học kỳ vừa qua. Trong quá trình học tập, thầy đã truyền đạt cho em vô vàn kiến thức hay và bổ ích, giúp em có được cơ sở lý thuyết vững vàng để em vượt qua bài báo cáo này dễ dàng hơn.

Bài báo cáo này cũng không thể tránh khỏi những thiếu sót và hạn chế, em rất mong quý thầy cô sẽ bỏ qua cho em và chỉ bảo thêm để giúp em có điều kiện bổ sung và làm tốt hơn trong những bài báo cáo sau này.

Em xin kính chúc quý thầy, quý cô và quý nhà trường luôn mạnh khỏe, hạnh phúc và ngày một thành công hơn trong sự nghiệp trồng người của mình.

Chúng em xin chân thành cảm ơn!

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

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của ThS 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 luận văn 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 luận văn 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)

Sương

Nguyễn Thị Diễm Sương

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

1. Vấn đề nghiên cứu:

Bài làm gồm có 2 chương:

- Chương 1: Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy.
- Chương 2: Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

2. Các hướng tiếp cận

- Lý thuyết.
- Thực hành.

3. Cách giải quyết vấn đề

Xem lại những nội dung đã học qua slide bài giảng, các kiến thức được ghi chép lại trong quá trình học và nghiên cứu thêm các video bài giảng trên mạng. Vận dụng chúng vào để giải quyết các nội dung trong đề thi.

4. Một số kết quả đạt được

Ôn lại được những kiến thức đã học, nắm vững các lý thuyết và phương pháp làm bài môn Nhập môn học máy. Rèn luyện tư duy logic cho việc học tập các môn học sau.

MỤC LỤC

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

DANH MỤC BẢNG BIỂU

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

1.1 Optimizer là gì, tại sao phải dùng nó?

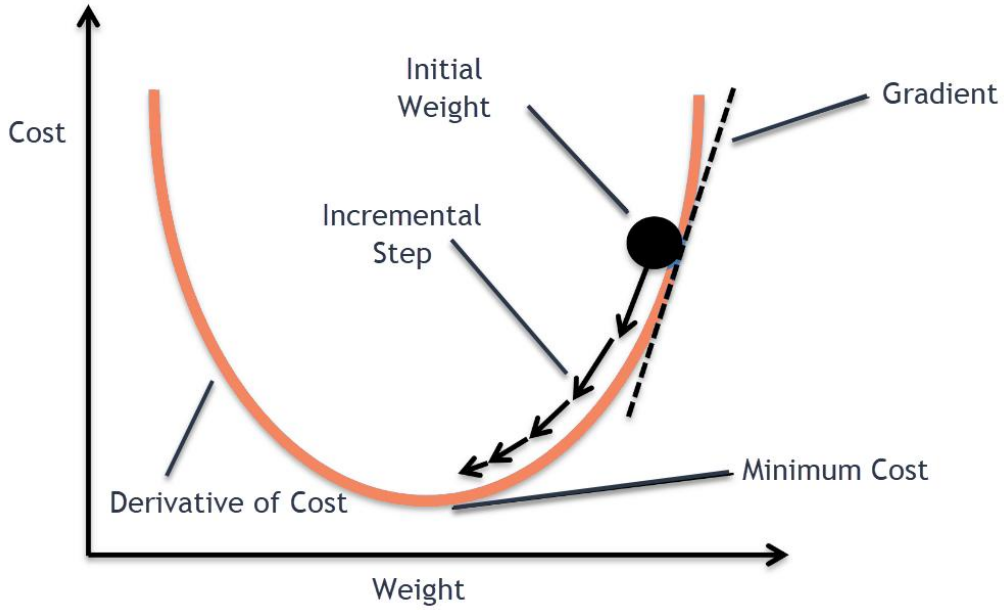
Trong lĩnh vực học máy, "optimizer" (trình tối ưu) là một thành phần quan trọng của quá trình huấn luyện mô hình máy học. Nhiệm vụ chính của optimizer là tối ưu hóa hàm mất mát bằng cách điều chỉnh các trọng số của mô hình để giảm thiểu giá trị của hàm mất mát.

1.2 Các thuật toán tối ưu

1.2.1 Gradient Descent (GD)

Gradient Descent (GD) là thuật toán tìm tối ưu chung cho các hàm số. Ý tưởng chung của GD là điều chỉnh các tham số để lặp đi lặp lại thông qua mỗi dữ liệu huấn luyện để giảm thiểu hàm chi phí.

Giả sử bạn bị lạc trên 1 ngọn núi và trong sương mù dày đặc, bạn chỉ có thể cảm thấy độ dốc của mặt đất dưới chân bạn. Một cách tốt nhất để nhanh chóng xuống chân núi là xuống dốc theo hướng dốc nhất. Đây chính là ý tưởng của Gradient Descent thực hiện, tại mỗi điểm của hàm số, nó sẽ xác định độ dốc sau đó đi ngược lại với hướng của độ dốc đến khi nào độ dốc tại chỗ đó bằng 0 (cực tiểu).



Hình 1. 1 Mô phỏng thuật toán Gradient Descent

Gradient Descent là một thuật toán tối ưu lặp (iterative optimization algorithm) được sử dụng trong các bài toán Machine Learning và Deep Learning (thường là các bài toán tối ưu lồi — Convex Optimization) với mục tiêu là tìm một tập các biến nội tại (internal parameters) cho việc tối ưu models. Trong đó:

- Gradient: là tỷ lệ độ nghiêng của đường dốc (rate of inclination or declination of a slope). Về mặt toán học, Gradient của một hàm số là đạo hàm của hàm số đó tương ứng với mỗi biến của hàm. Đối với hàm số đơn biến, chúng ta sử dụng khái niệm Derivative thay cho Gradient.
- Descent: là từ viết tắt của descending, nghĩa là giảm dần.

Gradient Descent có nhiều dạng khác nhau như Stochastic Gradient Descent (SGD), Mini-batch SDG. Nhưng về cơ bản thì đều được thực thi như sau:

- Khởi tạo biến nội tại.
- Đánh giá model dựa vào biến nội tại và hàm mất mát (Loss function).
- Cập nhật các biến nội tại theo hướng tối ưu hàm mất mát (finding optimal points).

- Lặp lại bước 2, 3 cho tới khi thỏa điều kiện dừng.
- Công thức cập nhật cho GD có thể được viết là:

$$\theta^{(nextstep)} = \theta - \eta \nabla_\theta$$

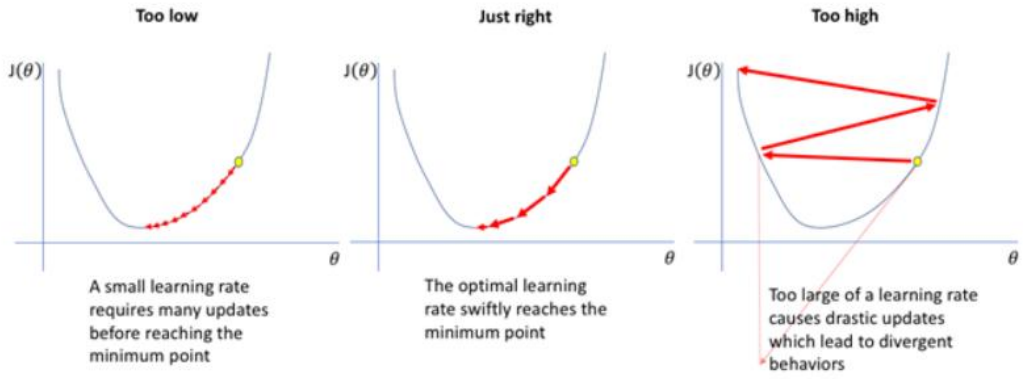
Trong đó θ là tập các biến cần cập nhật, η là tốc độ học (learning rate), $\nabla_\theta f(\theta)$ là Gradient của hàm mất mát f theo tập θ.

1.2.1.1 Learning rate:

Có 1 tham số quan trọng trong Gradient Descent đó là giá trị độ lớn của mỗi lần di chuyển (giống như độ dài sải chân khi bạn leo xuống dốc).

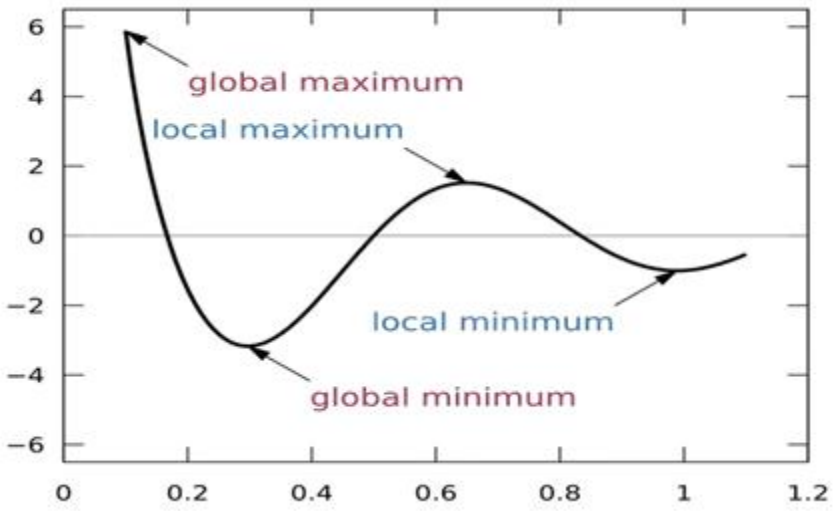
Tham số này được gọi là learning rate (tốc độ học). Nếu learning rate quá nhỏ, thuật toán sẽ phải thực hiện nhiều bước để hội tụ và sẽ mất nhiều thời gian.

Tuy nhiên nếu learning rate quá lớn sẽ khiến thuật toán đi qua cực tiểu, và vượt hẳn ra ngoài khiến thuật toán không thể hội tụ được.



Hình 1. 2 Sự ảnh hưởng của learning rate đến mô hình

Trong thực tế, không phải hàm số nào cũng chỉ có 1 cực tiểu. Ta sẽ có khái niệm cực tiểu cục bộ và cực tiểu toàn cục. Hiểu nôm na nó giống như các hố hoặc các tảng đá ở trên núi khi bạn đang leo xuống núi. Lúc này việc tìm ra cực tiểu sẽ trở nên khó khăn hơn. Xem hình sau để biết chi tiết:



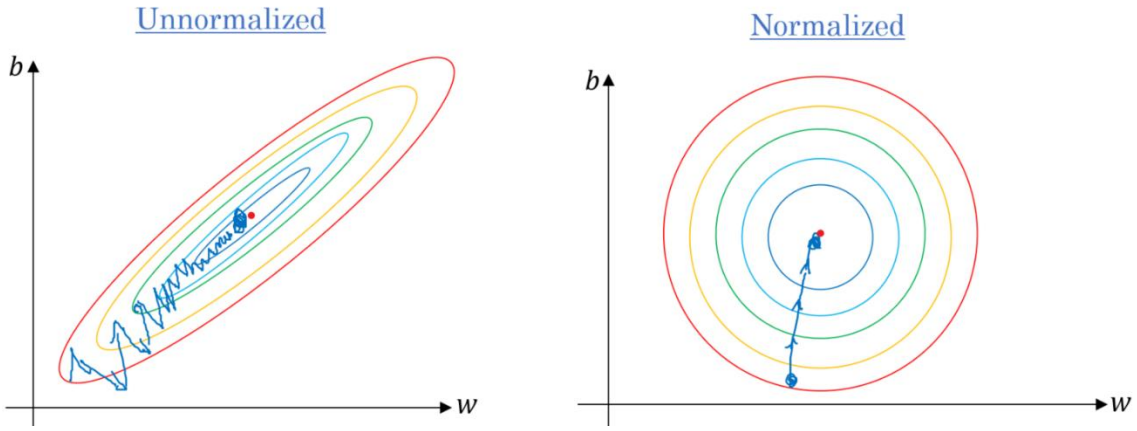
Hình 1. 3 Cực trị hàm số.

Sẽ có 2 vấn đề lúc này đối với GD:

Điểm xuất phát có thể ở bên trái hoặc bên phải, nếu xuất phát từ bên trái, thuật toán sẽ hội tụ ở local minimum (cực tiểu địa phương) mà không đi đến được global minium (cực tiểu toàn cục).

Hoặc nếu xuất phát từ bên phải sẽ phải mất nhiều thời gian để vượt qua Plateau để đến được global minimum và nếu kết thúc thuật toán quá sớm thì sẽ không đến được global minimum.

Trên thực tế, hàm chi phí có dạng đồ thị giống chiếc bát, nếu các feature (đặc điểm của đầu vào - thành phần của vector X) có cùng phạm vi giá trị, thì miệng bát sẽ tròn và để GD đi xuống đáy bát sẽ nhanh hơn. Nếu các feature khác phạm vi giá trị thì miệng bát sẽ bị kéo dài ra và việc đi xuống đáy bát sẽ tốn thời gian hơn. Đây là lý do vì sao các feature của vector đầu vào X cần phải được scaling (căn chỉnh).



Hình 1. 4 GD có scaling và không có scaling

Có thể thấy, ở bên phải thuật toán Gradient Descent đi thẳng về điểm tối thiểu, do đó nhanh chóng đạt được cực tiểu toàn cục, trong khi bên trái, nó đi theo hướng gần như trực giao với hướng về cực thiểu toàn cục, vì vậy nó kết thúc bằng 1 hành trình dài xuống một 1 mặt gần như bằng phẳng. Cuối cùng nó sẽ đạt đến mức cực tiểu, nhưng sẽ mất nhiều thời gian.

1.2.1.2 Batch Gradient Descent

Để thực hiện thuật toán Gradient Descent, chúng ta phải tìm được đạo hàm của hàm chi phí ảnh hưởng đến từng tham số của mô hình θ_j. Nói cách khác, cần phải xác định được giá trị hàm chi phí thay đổi như thế nếu thay đổi θ_j. Này được gọi là đạo hàm riêng (partial derivative).

Biểu thức sau sẽ dùng để tính đạo hàm riêng của hàm chi phí cho tham số θ_j, được ký hiệu là $\frac{\delta}{\delta \theta_j}$ MSE(θ):

$$\frac{\delta}{\delta\theta_j} MSE(\theta) = \frac{2}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_j^{(i)}$$

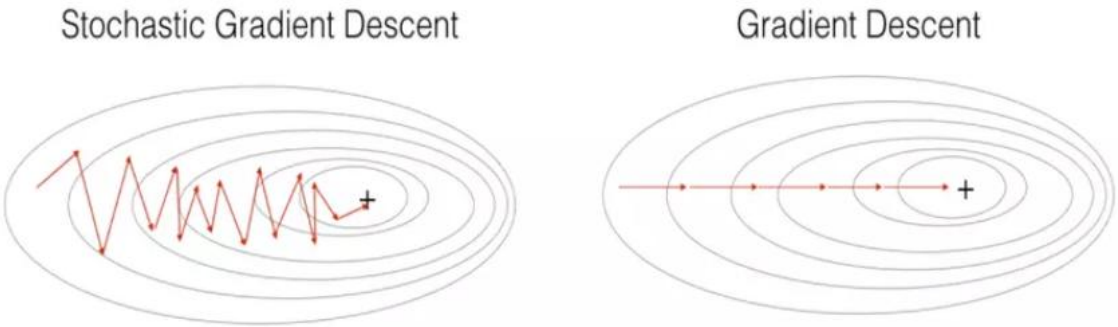
Thay vì tính từng đạo hàm thành phần, bạn có thể sử dụng công thức sau để tính tất cả trong một bước. Vector tốc độ, ký hiệu $\nabla_\theta \text{MSE}(\theta)$ là đạo hàm riêng (vector tốc độ) cho các tham số θ của mô hình.

Khi chúng ta có vector độ dốc và vị trí hiện tại, chúng ta chỉ cần đi ngược lại với vector độ dốc. Nghĩa là ta phải trừ đi θ 1 giá trị là $\nabla_\theta \text{MSE}(\theta)$. Lúc này ta sẽ sử dụng tham số learning rate η để xác định giá trị của bước xuống dốc bằng cách nhân vào.

$$\theta^{(nextstep)} = \theta - \eta \nabla_\theta MSE(\theta)$$

1.2.2 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.



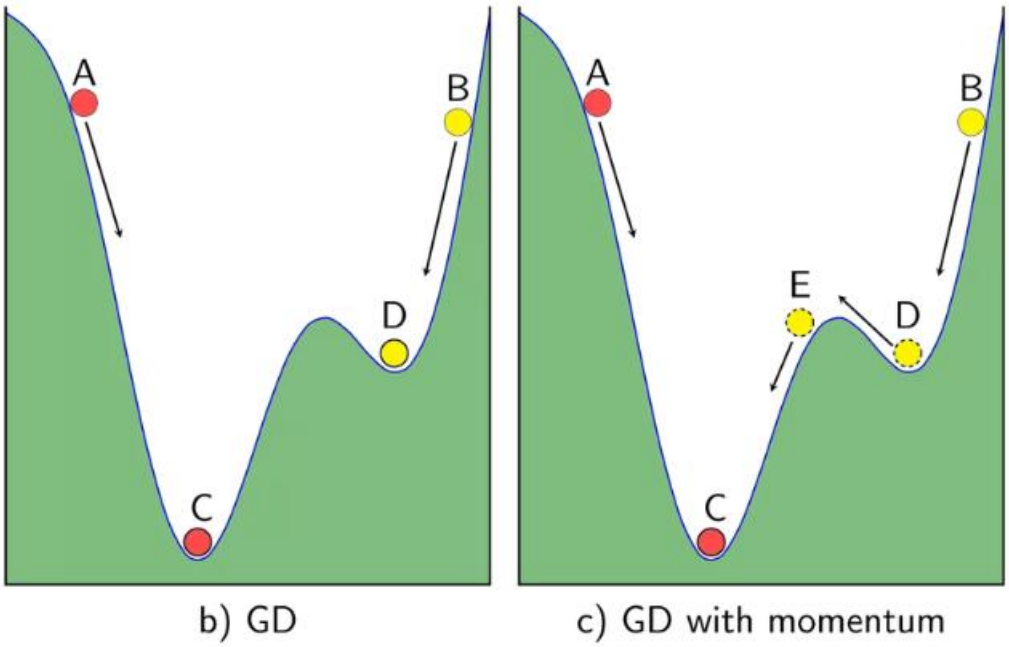
Hình 1. 5 Hình minh họa Stochastic Gradient Descent và Gradient Descent

Nhìn vào 2 hình trên, ta thấy SGD có đường đi khá là zig zắc , không mượt như GD. Dễ hiểu điều đó vì 1 điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu. Ở đây, GD có hạn chế đối với cơ sở dữ liệu lớn (vài triệu dữ liệu) thì việc tính toán đạo hàm trên toàn bộ dữ liệu qua mỗi vòng lặp trở nên cồng kềnh. Bên cạnh đó GD không phù hợp với online learning.

Online learning là khi dữ liệu cập nhật liên tục (ví dụ như thêm người dùng đăng kí) thì mỗi lần thêm dữ liệu ta phải tính lại đạo hàm trên toàn bộ dữ liệu => 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.

1.2.3 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

1.2.4 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.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

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.

1.2.5 RMSprop

Thuật toán RMSProp rất giống với Adagrad ở chỗ cả hai đều sử dụng bình phương của gradient để thay đổi tỉ lệ hệ số. RMSProp có điểm chung với phương pháp động lượng là chúng đều sử dụng trung bình rò rỉ. Tuy nhiên, RMSProp sử dụng kỹ thuật này để điều chỉnh tiền điều kiện theo hệ số.

Cách thức thuật toán hoạt động:

1. Khởi tạo tham số: Bắt đầu với việc khởi tạo các tham số của mô hình.
2. Khởi tạo giá trị đầu tiên: Tại thời điểm đầu tiên, tính toán giá trị trung bình của đạo hàm tại thời điểm đầu tiên:

   $E[g^2]_0 = 0$, trong đó g là đạo hàm của hàm mất mát theo từng tham số.
3. Cập nhật tham số: Tại mỗi bước lặp, tính toán giá trị trung bình có trọng số của bình phương của đạo hàm. Bước này giúp tạo ra một tham số tỷ lệ dựa trên độ lớn của đạo hàm:

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

   Trong đó β là hệ số độ đệm, thường được đặt giá trị khoảng 0.9
4. Cập nhật tham số theo learning rate tự điều chỉnh: Cập nhật tham số theo công thức:

   $$\text{Tham số mới} = \text{Tham số cũ} - \frac{\text{Tỷ lệ học}}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

   Trong đố $\in$ là một giá trị nhỏ (thường là 1e -8) được thêm vào trong mẫu số để tránh chia cho 0.

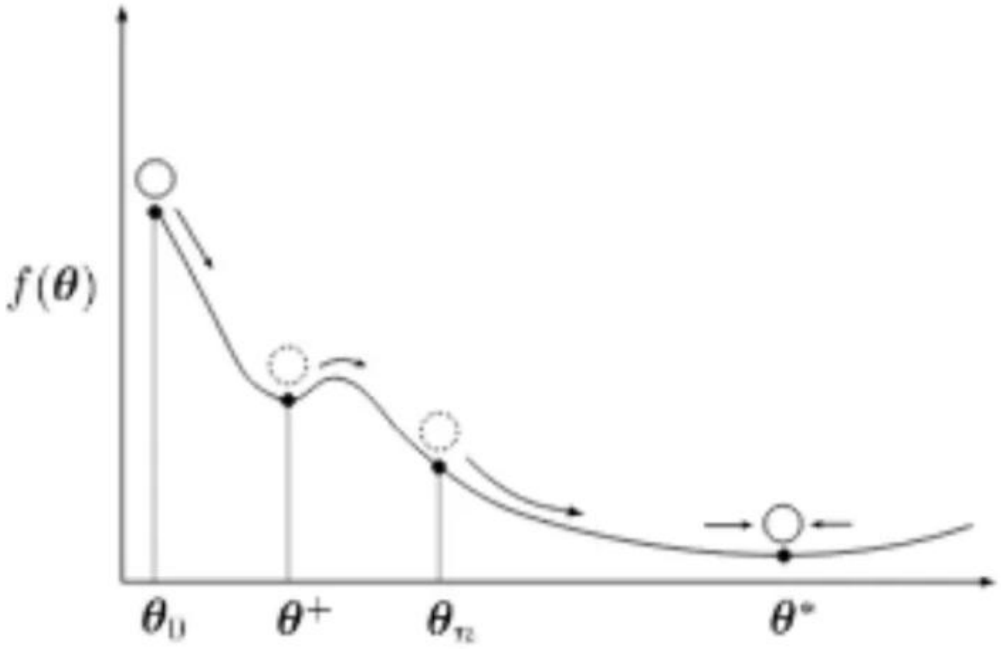
   Thuật toán RMSprop giúp giảm độ lớn của bước cập nhật dựa trên độ lớn của đạo hàm, đồng thời giữ cho quá trình học ổn định hơn và ít nhạy cảm hơn đối với biến động của tham số. Nó thường được ứng dụng hiệu quả trong nhiều bài toán học máy.

1.2.6 Adam

Adam (Adaptive Moment Estimation) là một thuật toán tối ưu hóa được sử dụng rộng rãi trong học máy để huấn luyện mô hình. Nó kết hợp hai khái niệm chính là Momentum và RMSprop để cải thiện quá trình tối ưu hóa.

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.



Cách Adam hoạt động:

1. Khởi tạo tham số: Bắt đầu với việc khởi tạo các tham số của mô hình, trọng số và bias.
2. Khởi tạo giá trị đầu tiên: Tại thời điểm đầu tiên (t=0) khởi tạo giá trị đầu tiên cho hai biến m_t (mometum) và v_t (bình phương của đạo hàm) là 0:
   $m_0 = 0$
   $v_o = 0$
3. Bước lặp (tính toán tại mỗi bước lặp):
   - Tính đạo hàm: Tính toán đạo hàm của hàm mất mát theo từng tham số g_t
   - Cập nhật m_t và v_t:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

Trong đó β_1 và β_2 là hệ số đệm (thường là 0.9 và 0.999).

- Hiệu chỉnh giá trị trung bình:

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

- Cập nhật tham số:

$$\text{Tham số mới} = \text{Tham số cũ} - \frac{\text{Tỷ lệ học}}{\sqrt{\hat{v}_t}+\epsilon} \cdot \hat{m}_t$$

Trong đó ∈ là một giá trị nhỏ (thường là 1e-8) được thêm vào mẫu số để tránh chia cho 0.

4. Lặp lại bước 3 cho đến khi điều kiện dừng được đáp ứng, chẳng hạn như số lần lặp tối đa, độ lớn của g_t nhỏ hone một ngưỡng, hoặc giảm thiểu hàm mất mát đủ.

Adam giúp tối ưu hóa mô hình một cách hiệu quả và tự điều chỉnh tốc độ học dựa trên độ lớn của đạo hàm và quá trình cập nhật trọng số trước đó. Nó thường được ưa chuộng trong nhiều bài toán học máy do khả năng linh hoạt và hiệu suất cao.

1.3 So sánh các phương pháp Optimizer trong huấn luyện mô hình học máy:

Thuật toán	Ưu điểm	Nhược điểm
Gradient Descent (GD)	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	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

	vòng lặp.	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
Stochastic Gradient Descent (SGD)	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.	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,..
Momentum	Giảm dao động: Hỗ trợ giảm dao động của quá trình cập nhật trọng số, giúp mô hình hội tụ nhanh hơn. Quá trình hội tụ nhanh hơn: Momentum giúp mô hình vượt qua các vùng	Overshooting: Có thể "đi quá" mục tiêu và tăng khả năng overshooting, đặc biệt trên các địa hình núi non của hàm mất mát. Yếu với một số vấn đề: Momentum không phải là lựa chọn tốt cho mọi loại

	phẳng trong không gian tham số, tăng tốc độ hội tụ.	vấn đề, đặc biệt là khi đối mặt với các hàm mất mát không lồi hoặc không liên tục. Không hiệu quả với giá trị ban đầu: Hiệu suất của Momentum có thể phụ thuộc nhiều vào giá trị khởi tạo ban đầu của động lượng, và một số giá trị khởi tạo không tốt có thể ảnh hưởng đến quá trình huấn luyện.
Adagrad	Một lợi 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.	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.
RMSProp	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	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

	thể dẫn tới bị đóng băng)	với RMSprop cho ra 1 thuật toán tối ưu Adam. Chúng ta sẽ trình bày nó trong phần sau.
Adam	Hiệu suất cao và nhanh chóng. Tự điều chỉnh tỷ lệ học. Hiệu quả với dữ liệu thưa. Dễ sử dụng.	Yếu với dữ liệu nhỏ hoặc mô hình đơn giản. Khả năng overshooting. Cần cẩn thận với siêu tham số.

Bảng 1. 1 So sánh các phương pháp Optimizer

CHƯƠNG 2: TÌM HIỂU VỀ CONTINUAL LEARNING VÀ TEST PRODUCTION KHI XÂY DỰNG MỘT GIẢI PHÁP HỌC MÁY ĐỂ GIẢI QUYẾT BÀI TOÁN

2.1 Continual Learning là gì?

Continual Learning (CML) là một quá trình trong đó một mô hình học từ các luồng dữ liệu mới mà không cần đào tạo lại.

Trái ngược với các phương pháp tiếp cận truyền thống, trong đó các mô hình được đào tạo trên một tập dữ liệu tĩnh, được triển khai và đào tạo lại định kỳ, các mô hình học liên tục cập nhật lặp đi lặp lại các tham số của chúng để phản ánh các phân phối mới trong dữ liệu.

Trong quy trình sau, mô hình tự cải thiện bằng cách học hỏi từ lần lặp mới nhất và cập nhật kiến thức của nó khi có dữ liệu mới. Vòng đời của mô hình học tập liên tục cho phép các mô hình duy trì tính phù hợp theo thời gian nhờ chất lượng năng động vốn có của chúng.

2.1.1 Các loại Continual Learning:

Có nhiều phương pháp học máy liên tục để lập mô hình:

Incremental learning

transfer learning

Lifelong learning.

Giống như tất cả dữ liệu, việc lựa chọn cách tiếp cận không phải là trắng đen. Thay vào đó, nó phụ thuộc vào dữ liệu, kiến trúc mô hình, hiệu suất mong muốn, độ phức tạp của nhiệm vụ và tài nguyên tính toán sẵn có. Thông thường, sự kết hợp của các phương pháp tiếp cận được thực hiện để nâng cao khả năng học tập.

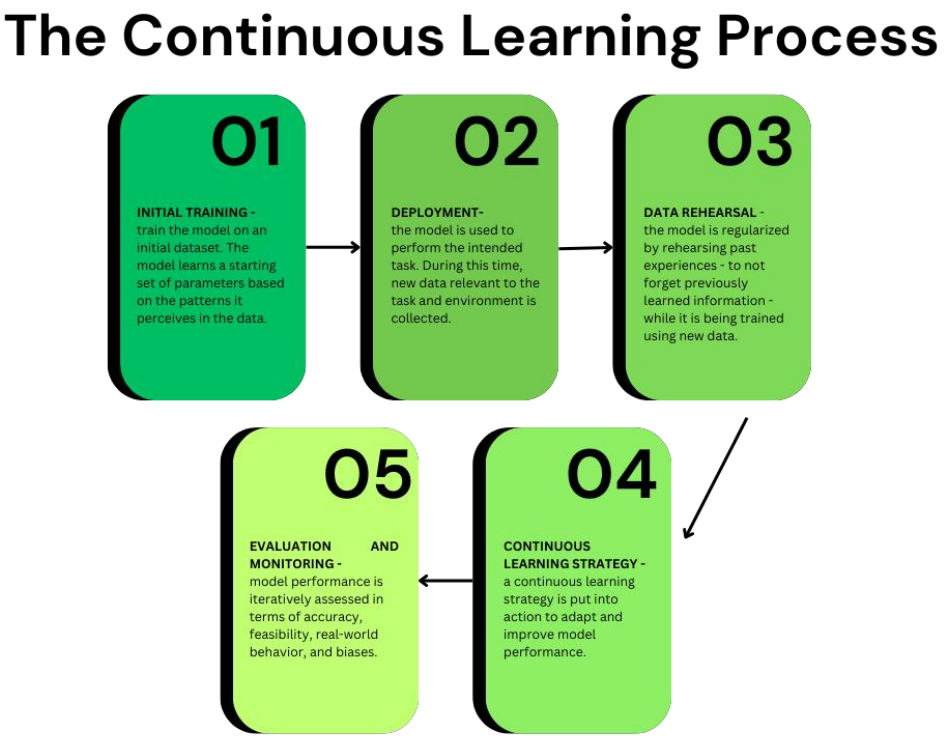
2.1.2 Quá trình Continual Learning:

Continual Learning là một sự phát triển của mô hình học máy truyền thống. Do đó, nó liên quan đến nhiều nguyên tắc lập mô hình giống nhau: tiền xử lý, lựa chọn mô hình, điều chỉnh siêu tham số, đào tạo, triển khai và giám sát.

Hai bước bổ sung cần thiết trong quá trình học tập liên tục:

+ Diễn tập dữ liệu
+ Thực hiện chiến lược học tập liên tục.

Các bước này nhằm đảm bảo rằng mô hình đang học từ các luồng dữ liệu mới một cách hiệu quả dựa trên ứng dụng và bối cảnh của nhiệm vụ dữ liệu.



Hình 2. 1 Sơ đồ quy trình Continual Learning trong học máy

2.1.3 Hạn chế của Continual Learning:

– Từ quan điểm hoạt động, một thách thức là: Các phương pháp học liên tục, tuy hiệu quả nhưng cũng có xu hướng phức tạp hơn về mặt tính toán so với các phương pháp truyền thống vì mô hình cần phải thích ứng nhất quán với dữ liệu

mới. Sự phức tạp nói trên thường dẫn đến chi phí kinh tế cao hơn vì nó đòi hỏi nhiều tài nguyên dữ liệu, con người và máy tính hơn.

- Từ góc độ mô hình hóa, một số nhược điểm là:
  + Quản lý mô hình: Mỗi khi các tham số của mô hình cập nhật dựa trên dữ liệu mới, một mô hình mới sẽ được hình thành. Do đó, cách tiếp cận học tập liên tục có thể tạo ra một số lượng lớn các mô hình, làm phức tạp việc xác định những mô hình hoạt động tốt nhất.
  + Dữ liệu trôi dạt: Để phương pháp học tập liên tục có giá trị, chúng ta phải xử lý một khối lượng lớn dữ liệu mới. Tuy nhiên, mô hình như vậy có nguy cơ mất khả năng dự đoán nếu phân bổ tính năng thay đổi đột ngột. Tìm hiểu thêm về sự trôi dạt dữ liệu trong một bài viết riêng.

2.1.4 Ứng dụng của Continual Learning:

An ninh mạng: Các phương pháp học tập liên tục được triển khai để đảm bảo giám sát liên tục trong cơ sở hạ tầng bảo mật CNTT. Chúng là chìa khóa trong việc phát hiện hành vi lừa đảo, xâm nhập mạng và thư rác, cùng các hoạt động liên quan đến bảo mật khác.

Chăm sóc sức khỏe: Do bản chất ngày càng phát triển của bệnh tật, các phương pháp học tập liên tục được sử dụng trong các lĩnh vực chăm sóc sức khỏe khác nhau để tăng cường quy trình chẩn đoán xung quanh việc chẩn đoán bệnh. Các chuyên ngành như ung thư và X quang đã là những người đi đầu trong việc khám phá AI học tập liên tục để đạt được mục tiêu này.

Người máy: Học tập liên tục đã được sử dụng để nâng cao khả năng thích ứng và hiệu suất của robot trong các môi trường khác nhau, cho phép chúng tối ưu hóa hành động của mình trong các điều kiện thay đổi dựa trên kinh nghiệm trong quá khứ và mới.

2.2 Tìm hiểu về Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

Trong mô hình học máy, "Testing in Production" có thể ám chỉ các chiến lược kiểm thử mà các nhà phát triển và chuyên gia học máy sử dụng để đảm bảo tính hiệu quả và độ chính xác của mô hình trong môi trường sản xuất.

Dưới đây là một số chiến lược thường được áp dụng:

- A/B Testing (Kiểm thử A/B): Sử dụng A/B testing để so sánh hiệu suất giữa một mô hình cũ và một mô hình mới trong môi trường sản xuất. Các nhóm người dùng được chia thành hai phần, một phần tiếp tục sử dụng mô hình cũ và một phần sử dụng mô hình mới. Điều này giúp đánh giá cách mà mô hình mới thực sự cải thiện so với mô hình hiện tại.
- Canary Testing (Kiểm thử Canary): Tương tự như trong phần mềm thông thường, chiến lược Canary Testing trong học máy liên quan đến việc triển khai một phiên bản mới của mô hình cho một phần nhỏ của dữ liệu hoặc người dùng trước khi triển khai toàn bộ mô hình mới.
- Shadow Testing (Kiểm thử Shadow): Tạo ra bản sao của lưu lượng dữ liệu thực tế và chạy nó qua mô hình mới để đánh giá hiệu suất mà không ảnh hưởng đến người dùng chính thức. Điều này giúp đảm bảo rằng mô hình mới hoạt động đúng trước khi áp dụng nó cho tất cả người dùng.
- Feature Toggles (Chuyển đổi Tính năng): Sử dụng chuyển đổi tính năng để tắt hoặc bật các tính năng của mô hình trong môi trường sản xuất mà không cần phải triển khai lại toàn bộ mô hình. Điều này giúp kiểm thử tính năng mới mà không làm gián đoạn hoạt động chung của hệ thống.

Mục tiêu chung của những chiến lược này là giảm thiểu rủi ro khi triển khai các cập nhật của mô hình trong môi trường sản xuất và đồng thời đảm bảo tính ổn định và độ chính xác của mô hình. Tuy nhiên, cần phải áp dụng chúng cẩn thận để tránh ảnh hưởng đến trải nghiệm người dùng và đảm bảo an toàn cho dữ liệu.

2.2.1 Ưu Điểm của Testing in Production trong Học Máy:

Thực Tế và Hiệu Quả: Kiểm thử trong môi trường sản xuất giúp đánh giá hiệu suất của mô hình dưới điều kiện thực tế, đồng thời giảm thiểu khoảng cách giữa môi trường kiểm thử và môi trường sản xuất.

Phát Hiện Lỗi Nhanh Chóng: Các chiến lược như Canary Testing và Shadow Testing giúp phát hiện lỗi và vấn đề hiệu suất ngay từ giai đoạn đầu mà không cần phải đợi đến khi mô hình được triển khai toàn bộ.

Kiểm Thử Tính Ứng Dụng và Tương Tác Người Dùng: Testing in Production cho phép kiểm tra cách mô hình ứng xử trong môi trường thực tế, bao gồm cả tương tác với người dùng cuối.

Cập Nhật Linh Hoạt: Sử dụng Feature Toggles giúp cập nhật mô hình một cách linh hoạt mà không làm gián đoạn quá nhiều hoạt động.

2.2.2 Nhược Điểm của Testing in Production trong Học Máy:

Rủi Ro Cho Trải Nghiệm Người Dùng: Việc kiểm thử trong môi trường sản xuất có thể tạo ra rủi ro và ảnh hưởng đến trải nghiệm người dùng nếu không được thực hiện cẩn thận.

An Toàn Dữ Liệu: Cần phải đảm bảo rằng quy trình kiểm thử không làm mất an toàn của dữ liệu người dùng, đặc biệt là khi sử dụng bản sao dữ liệu thực tế.

Khó Khăn Trong Việc Kiểm Soát Biến Đổi: Sự linh hoạt của Testing in Production có thể tạo ra khó khăn trong việc kiểm soát các biến đổi và hiểu rõ về tác động của chúng lên mô hình.

Yêu Cầu Sự Cẩn Thận và Quản Lý: Để triển khai thành công chiến lược kiểm thử trong môi trường sản xuất, cần phải có quản lý và cẩn thận cao, đặc biệt là trong việc xử lý dữ liệu nhạy cảm.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Optimizer- Hiểu sâu về các thuật toán tối ưu (GD,SGD,Adam,..)
https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8

[2] Ưu điểm của AI Adam Optimizer trong nâng cao hiệu suất mô hình học sâu
https://funix.edu.vn/chia-se-kien-thuc/ai-adam-optimizer-nang-cao-hieu-suat-cua-mo-hinh-hoc-sau/

Tiếng Anh

[1] What is Continuous Learning? Revolutionizing Machine Learning & Adaptability
https://www.datacamp.com/blog/what-is-continuous-learning

[2] Machine Learning in Production – Testing
https://applyingml.com/resources/testing-ml/