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

NGUYỄN QUANG LỢI - 52100910

BÁO CÁO CUỐI KỲ
NHẬP MÔN HỌC MÁY

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

NGUYỄN QUANG LỢI - 52100910

BÁO CÁO CUỐI KỲ
NHẬP MÔN HỌC MÁY

Người hướng dẫn
PGS.TS. Lê Anh Cường

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến Trường Đại học Tôn Đức Thắng đã đưa môn học Nhập môn học máy vào chương trình giảng dạy. Xin cảm ơn khoa Công nghệ thông tin đã cung cấp những tài liệu học tập đầy đủ, chi tiết. Xin gửi lời cảm ơn chân thành PGS.TS. Lê Anh Cường, người đã giảng dạy chúng em trong suốt quá trình tham gia môn học. Chúng em rất biết ơn những bài tập thầy giao đã giúp chúng em củng cố kiến thức, đã giải đáp mọi thắc mắc của chúng em.

TP. Hồ Chí Minh, ngày 20 tháng 12 năm 2023

Tác giả

(Ký tên và ghi rõ họ tên)

Lợi

Nguyễn Quang Lợi

CÔNG TRÌNH 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 PGS.TS. 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 Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 20 tháng 12 năm 2023

Tác giả

(Ký tên và ghi rõ họ tên)

Lợi

Nguyễn Quang Lợi

TÓM TẮT

Chương 1 ta sẽ tìm hiểu về các phương pháp Optimizer trong huấn luyện mô hình học máy.

Chương 2 ta sẽ tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

MỤC LỤC

DANH MỤC HÌNH VẼ vi

DANH MỤC BẢNG BIỂU vii

DANH MỤC CÁC CHỮ VIẾT TẮT viii

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

1.1 Optimizer là gì 1

1.2 Các loại optimizer 1

1.2.1 Gradient Decent 1

1.2.2 Stochastic Gradient Descent (SGD) 3

1.2.3 Momentum 3

1.2.4 AdaGrad(Adaptive Gradient Descent) 5

1.2.5 AdaDelta 6

1.2.6 RMS-Prop (Root Mean Square Propagation) 8

1.2.7 Adam(Adaptive Moment Estimation) 9

1.3 Phân tích so sánh 11

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 14

2.1 Continual Learning 14

2.1.1 Khái niệm 14

2.1.2 Quá trình Continual Learning 17

2.1.3 Các loại Continual Learning 17

2.1.4 Continual Learning hoạt động như thế nào trong học máy 20

2.1.5 Ưu điểm và hạn chế của quá trình Continual Learning *22*

2.1.6 Ứng dụng của Continual Learning *24*

2.2 Test Production 24

2.2.1 Giới thiệu *24*

2.2.2 Quá trình thử nghiệm *25*

2.2.3 Các loại testing production *28*

TÀI LIỆU THAM KHẢO 30

DANH MỤC HÌNH VẼ

Hình 1.1: Gradient Decent 2
Hình 1.2: Các loại Gradient Decent 2
Hình 2.1: Định nghĩa CI, CT, CD và CML 15
Hình 2.2: Sơ đồ quy trình học tập liên tục đến học máy 17
Hình 2.3: Luồng thử nghiệm học máy 25
Hình 2.4: Post training test 27
Hình 2.5: Stage Test 28
Hình 2.6: API Testing 28

DANH MỤC BẢNG BIỂU

Bảng 1.1: Phân tích so sánh các Optimizer 11

DANH MỤC CÁC CHỮ VIẾT TẮT

SGD	Stochastic Gradient Descent
Adagrad	Adaptive Gradient Algorithm
Adadelta	Adaptive Delta
RMSProp	Root Mean Square Propagation
Adam	Adaptive Moment Estimation

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

1.1 Optimizer là gì

Optimizer là các thuật toán hoặc phương pháp được sử dụng để giảm thiểu hàm mất mát (loss function). Optimizer là các hàm toán học phụ thuộc vào các tham số có thể học được của mô hình, tức là trọng số (weight) và độ lệch (bias). Optimizer giúp ta biết cách thay đổi trọng số và tốc độ học (learning rate) của mạng neural để giảm tổn thất.

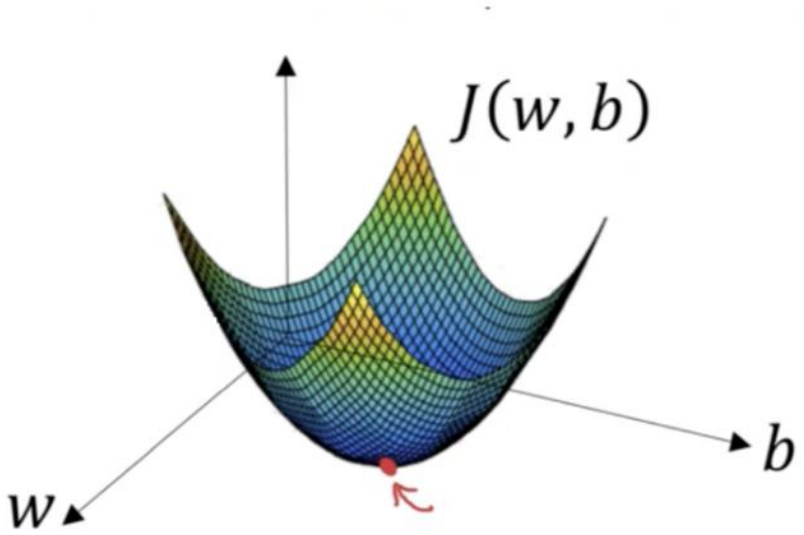
1.2 Các loại optimizer

1.2.1 Gradient Decent

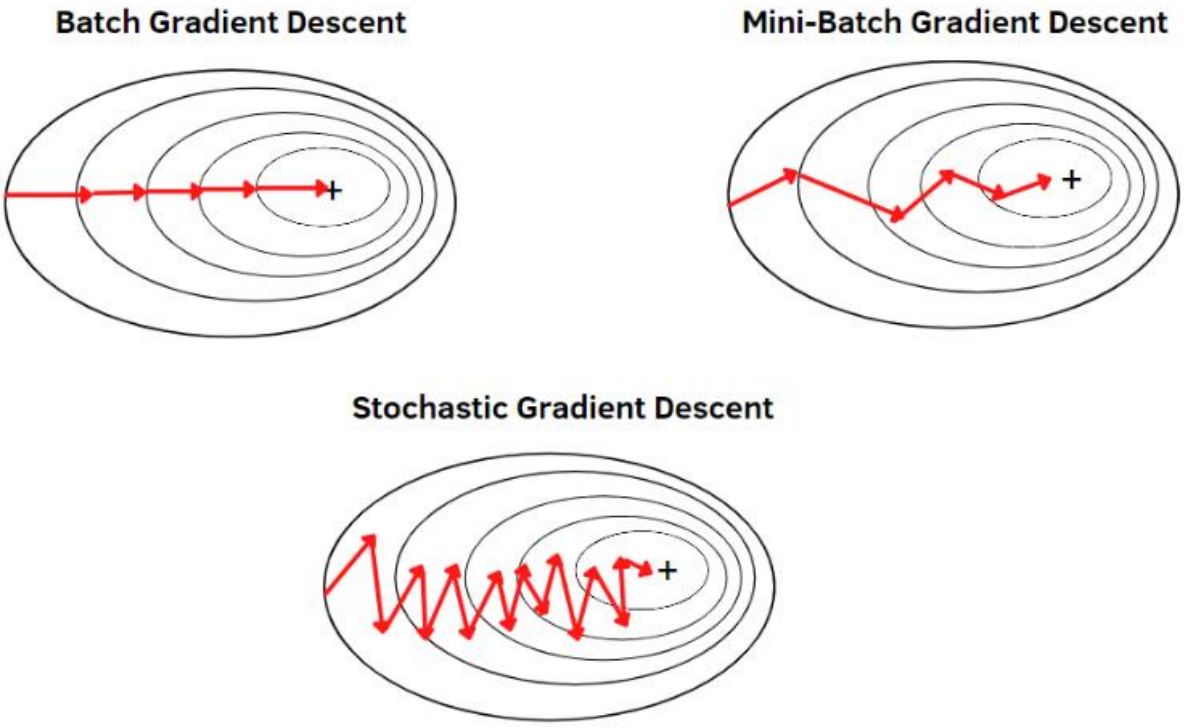
Gradient Descent là thuật toán tối ưu hóa cơ bản nhất nhưng được sử dụng nhiều nhất. Nó được sử dụng rất nhiều trong các thuật toán phân loại và hồi quy tuyến tính. Lan truyền ngược trong mạng nơ-ron cũng sử dụng thuật toán Gradient Descent.

Gradient descent là một thuật toán tối ưu hóa dựa trên một hàm lồi (convex function) và điều chỉnh các tham số của nó một cách lặp lại để giảm thiểu một hàm cho trước đến giá trị tối thiểu cục bộ. Gradient Descent giảm thiểu một hàm mất mát một cách lặp lại bằng cách di chuyển theo hướng ngược với hướng tăng nhanh nhất. Nó phụ thuộc vào đạo hàm của hàm mất mát để tìm giá trị nhỏ nhất. Trong Machine Learning, Gradient Descent sử dụng dữ liệu từ toàn bộ bộ dữ liệu huấn luyện để tính đạo hàm của hàm chi phí đối với các tham số, điều này đòi hỏi một lượng lớn bộ nhớ và làm chậm quá trình.

$$\mathrm{W}_{\mathrm{new}} = \left(\mathrm{W}_{\mathrm{old}} - \alpha \times \frac{\partial(Loss)}{\partial(\mathrm{W}_{\mathrm{old}})}\right) \tag{1.1}$$



Hình 1.1: Gradient Decent



Hình 1.2: Các loại Gradient Decent

1.2.2 Stochastic Gradient Descent (SGD)

Giảm dần độ dốc ngẫu nhiên (SGD) là một kỹ thuật tối ưu hóa lặp lại được sử dụng rộng rãi trong học máy và học sâu. Đây là một biến thể của Gradient Descent cung cấp các cập nhật cho các tham số mô hình (trọng số) dựa trên độ dốc của hàm mất được tính toán trên một phần dữ liệu huấn luyện được chọn ngẫu nhiên, thay vì trên tập dữ liệu hoàn chỉnh.

Nguyên tắc cốt lõi của SGD là chọn một phần ngẫu nhiên nhỏ của dữ liệu huấn luyện, được gọi là mini-batch và tính toán độ dốc của hàm mất đối với các tham số mô hình chỉ sử dụng phần đó. Độ dốc này sau đó được sử dụng để cập nhật các tham số. Quy trình được tiếp tục với một mini-batch ngẫu nhiên mới cho đến khi thuật toán hội tụ hoặc đạt được điều kiện dừng xác định trước.

SGD cung cấp nhiều lợi thế khác nhau so với Gradient Decent truyền thống, chẳng hạn như hội tụ nhanh hơn và nhu cầu bộ nhớ thấp hơn, đặc biệt đối với các bộ dữ liệu lớn. Nó cũng có khả năng phục hồi tốt hơn đối với dữ liệu nhiễu và không cố định, đồng thời có thể thoát khỏi mức tối thiểu cục bộ (local minima). Tuy nhiên, nó có thể cần nhiều lần lặp hơn để hội tụ hơn là Gradient Decent và tốc độ học (learning rate) cần được hiệu chỉnh cẩn thận để đảm bảo sự hội tụ.

Giảm dần độ dốc ngẫu nhiên là một kỹ thuật tối ưu hóa lặp lại sử dụng các nhóm dữ liệu nhỏ để hình thành kỳ vọng về độ dốc thay vì độ dốc đầy đủ bằng cách sử dụng tất cả dữ liệu có sẵn. Đó là đối với trọng số (W) và hàm mất mát L chúng ta có:

$$W_{t+1} = W_t - n\,\nabla_w\, L_{W(t)} \tag{1.3}$$

Trong đó n là tốc độ học (learning rate). SGD giảm sự dư thừa so với phương pháp giảm độ dốc hàng loạt - tính toán lại độ dốc cho các ví dụ tương tự trước mỗi lần cập nhật tham số - vì vậy nó thường nhanh hơn nhiều.

1.2.3 Momentum

Momentum là một chiến lược tối ưu hóa được sử dụng trong học máy và học sâu để đẩy nhanh quá trình đào tạo mạng nơ-ron. Nó dựa trên khái niệm thêm một

phần của bản cập nhật trước đó vào bản cập nhật trọng số hiện tại trong suốt quá trình tối ưu hóa.

Trong tối ưu hóa momentum, độ dốc của hàm chi phí được tính theo từng trọng số trong mạng nơ-ron. Thay vì cập nhật trọng số trực tiếp dựa trên gradient, tối ưu hóa động lượng đưa ra một biến mới, gọi là số hạng động lượng (momentum term), được sử dụng để cập nhật trọng số. Thuật ngữ động lượng là trung bình động của các gradient và nó tích lũy các gradient trước đó để giúp tác động đến hướng tìm kiếm.

Thuật ngữ động lượng có thể được xem là vận tốc của bộ tối ưu hóa. Bộ tối ưu hóa thu được động lượng khi nó đi xuống dốc và dùng để làm giảm các dao động trong quá trình tối ưu hóa. Điều này có thể cho phép trình tối ưu hóa hội tụ nhanh hơn và đạt được mức tối thiểu cục bộ (local minimum) tốt hơn.

Tối ưu hóa động lượng đặc biệt có lợi trong các trường hợp khi bối cảnh tối ưu hóa nhiễu hoặc khi độ dốc thay đổi nhanh. Nó cũng có thể giúp làm trơn tru quá trình tối ưu hóa và tránh trình tối ưu hóa bị mắc kẹt trong mức tối thiểu cục bộ.

Cách mà momentum hoạt động:

Động lực tối ưu hóa giống như một quả bóng lăn xuống dốc. Trong khi độ dốc giảm dần cập nhật các tham số dựa trên độ dốc hiện tại, động lượng sẽ thêm một phần của bản cập nhật trước đó vào phần hiện tại. Điều này giúp trình tối ưu hóa duy trì theo cùng một hướng, giảm dao động và tránh bị kẹt ở cực tiểu cục bộ, giải quyết các nhược điểm của phương pháp giảm độ dốc truyền thống. Quy tắc cập nhật động lượng có thể được viết như sau:

$$v_t = \beta v_{t-1} + (1-\beta)\,\nabla_\theta J(\theta) \tag{1.4}$$

$$\theta = \theta - \alpha v_t$$

- $J(\theta)$ là hàm chi phí..
- $\nabla_\theta J(\theta)$ là độ dốc của hàm chi phí đối với các tham số θ
- β là số hạng động lượng (momentum term)
- v là vectơ vận tốc hoặc động lượng

- α là tốc độ học

Thông thường, hệ số động lượng được đặt ở mức 0,9. Trình tối ưu hóa tính toán độ dốc của hàm chi phí tại mỗi lần lặp và cập nhật số hạng động lượng dưới dạng trung bình động có trọng số theo cấp số nhân của các độ dốc trước đó. Các tham số sau đó được cập nhật bằng cách loại bỏ số hạng động lượng nhân với tốc độ học.

Nhìn chung, động lượng là một chiến lược tối ưu hóa mạnh mẽ có thể hỗ trợ đẩy nhanh quá trình đào tạo mạng lưới nơ-ron sâu và tăng hiệu suất của chúng.

1.2.4 AdaGrad(Adaptive Gradient Descent)

Adagrad (Adaptive Gradient Descent) là một kỹ thuật tối ưu hóa được sử dụng trong học máy và học sâu để tối ưu hóa việc đào tạo mạng nơ-ron.

Phương pháp Adagrad điều chỉnh tốc độ học của từng tham số của mạng nơ-ron một cách thích ứng trong quá trình huấn luyện. Cụ thể, nó điều chỉnh tốc độ học của từng tham số dựa trên độ dốc thu được trước đó cho tham số đó. Nói cách khác, các tham số có độ dốc lớn sẽ có tốc độ học thấp hơn, trong khi những tham số có độ dốc nhỏ sẽ có tốc độ học lớn hơn. Điều này giúp ngăn tốc độ học giảm xuống quá nhanh đối với các tham số thường xuyên xảy ra và cho phép hội tụ quá trình đào tạo nhanh hơn.

Kỹ thuật Adagrad đặc biệt hiệu quả trong việc xử lý dữ liệu thưa thớt, khi các phần của đặc tính đầu vào có tần số thấp hoặc không có. Trong những trường hợp này, Adagrad có thể thay đổi tốc độ học của từng tham số một cách thích ứng, cho phép xử lý dữ liệu thưa thớt tốt hơn. Nhìn chung, Adagrad là một phương pháp tối ưu hóa mạnh mẽ có thể hỗ trợ tăng tốc độ đào tạo mạng lưới thần kinh sâu và nâng cao hiệu suất của chúng.

Cách thức hoạt động: Nguyên tắc chính của AdaGrad là chia tỷ lệ learning rate cho từng tham số theo tổng độ dốc bình phương quan sát được trong quá trình đào tạo. Các bước của thuật toán như sau:

- Khởi tạo biến

Khởi tạo các tham số θ và một hằng số nhỏ ϵ để tránh chia cho 0.

Khởi tạo tổng của biến gradient bình phương G bằng các số 0, có hình dạng giống như θ.

- Tính toán độ dốc

  Tính gradient của hàm mất mát đối với từng tham số, $\nabla\theta J(\theta)$

- Tích lũy gradient bình phương

  Cập nhật tổng gradient bình phương G cho mỗi tham số i:

  $G[i] += (\nabla\theta J(\theta[i]))^2$

- Cập nhật tham số

  Cập nhật từng tham số bằng tốc độ học thích ứng:

  $\theta[i] -= (\eta / (\sqrt{(G[i])} + \epsilon)) * \nabla\theta J(\theta[i])$

  η :tốc độ học

  $\nabla\theta J(\theta[i])$: gradient của hàm mất mát đối với tham số $\theta[i]$.

1.2.5 AdaDelta

Adadelta là một kỹ thuật tối ưu hóa được sử dụng trong học máy và học sâu để tối ưu hóa việc đào tạo mạng lưới thần kinh. Đây là một biến thể của phương pháp Adagrad khắc phục được một số nhược điểm của nó.

Phương pháp Adadelta sửa đổi tốc độ học của từng tham số theo cách tương tự như Adagrad, nhưng thay vì giữ lại tất cả các gradient trước đó, nó chỉ giữ lại mức trung bình động của các gradient bình phương. Điều này giúp giảm nhu cầu bộ nhớ của phương pháp.

Ngoài ra, Adadelta sử dụng một phương pháp gọi là "cập nhật delta" để thay đổi tốc độ học. Thay vì sử dụng tốc độ học tập đã đặt, Adadelta sử dụng tỷ lệ bình phương trung bình gốc (RMS) của các gradient trước đó và RMS của các bản cập nhật trước đây để mở rộng tốc độ học tập. Điều này giúp ngăn chặn hơn nữa tốc độ học tập giảm quá sớm đối với các đặc điểm thường xuyên xảy ra.

Giống như Adagrad, Adadelta đặc biệt hiệu quả trong việc xử lý dữ liệu thưa thớt, nhưng nó cũng có thể hoạt động tốt hơn trong trường hợp Adagrad có thể hội tụ quá nhanh.

AdaDelta là một kỹ thuật tối ưu hóa ngẫu nhiên cho phép áp dụng phương pháp tốc độ học theo chiều cho SGD. Nó là một phần mở rộng của Adagrad nhằm tìm cách giảm tốc độ học tập giảm dần và đơn điệu. Thay vì tích lũy tất cả các gradient bình phương trong quá khứ, Adadelta giới hạn cửa sổ của các gradient trong quá khứ được tích lũy ở một kích thước cố định w.

Thay vì lưu trữ không hiệu quả w gradient bình phương trước đó, tổng gradient được xác định đệ quy là trung bình phân rã của tất cả gradient bình phương trước đó. Khi đó, mức trung bình đang chạy ở bước thời gian chỉ phụ thuộc vào độ dốc trung bình trước đó và hiện tại:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g^2 t \tag{1.5}$$

- $E[g^2]_t$: Trung bình chạy của gradient bình phương tại thời điểm t.
- γ: Hệ số suy giảm hoặc hệ số làm mịn, giá trị từ 0 đến 1 xác định trọng số cho giá trị trung bình chạy trước đó.
- $\gamma E[g^2]_{t-1}$: Trung bình chạy của các gradient bình phương ở bước thời gian trước đó (t−1).
- g_t: Độ dốc của hàm chi phí đối với các tham số tại thời điểm t.

Thông thường γ được đặt ở khoảng 0,9. Viết lại cập nhật SGD theo vectơ cập nhật tham số:

$$\Delta\theta_t = -n * g_{t,i} \tag{1.6}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

AdaDelta có dạng:

$$\Delta\theta_t = \frac{-n\, g_t}{\sqrt{E[g^2]_t + \epsilon}} \tag{1.7}$$

- $\Delta\theta_t$: Cập nhật tham số tại thời điểm t.
- n: Tốc độ học, một đại lượng vô hướng dương xác định kích thước bước trong không gian tham số.

- g_t: Gradient của hàm chi phí đối với các tham số tại thời điểm t
- $E[g^2]_t$: Trung bình chạy của gradient bình phương tại thời điểm t
- ϵ: Hằng số nhỏ được thêm vào để ổn định số học nhằm tránh chia cho 0 ở mẫu số.

Ưu điểm chính của AdaDelta là chúng ta không cần đặt tốc độ học mặc định.

Nhìn chung, Adadelta là một kỹ thuật tối ưu hóa mạnh mẽ có thể hỗ trợ đẩy nhanh quá trình đào tạo mạng lưới thần kinh sâu và tăng hiệu suất của chúng, đồng thời giải quyết một số nhược điểm của Adagrad.

1.2.6 RMS-Prop (Root Mean Square Propagation)

RMSProp (Root Mean Square Propagation) là một phương pháp tối ưu hóa được sử dụng trong học máy và học sâu để tối ưu hóa việc đào tạo mạng lưới thần kinh.

Giống như Adagrad và Adadelta, RMSProp sửa đổi tốc độ học của từng tham số trong suốt quá trình đào tạo. Tuy nhiên, thay vì thu thập tất cả các gradient trước đó như Adagrad, RMSProp tính toán mức trung bình động của các gradient bình phương. Điều này giúp thuật toán sửa đổi tốc độ học chậm hơn và ngăn tốc độ học giảm xuống quá sớm.

Kỹ thuật RMSProp cũng sử dụng hệ số phân rã để hạn chế ảnh hưởng của độ dốc trước đó đến tốc độ học. Hệ số phân rã này cho phép thuật toán tăng trọng số cho các gradient gần đây và ít trọng số hơn cho các gradient cũ hơn.

Một trong những ưu điểm chính của RMSProp so với Adagrad là nó có thể xử lý các mục tiêu không cố định, trong đó chức năng cơ bản mà mạng nơ-ron đang cố gắng bắt chước thay đổi theo thời gian. Trong một số trường hợp nhất định, Adagrad có thể hội tụ quá nhanh, trong khi RMSProp có thể điều chỉnh tốc độ học theo hàm mục tiêu thay đổi.

Đây là cách RMSProp hoạt động:

- Khởi tạo: Khởi tạo một biến trung bình đang chạy về 0, trong đó là độ dốc của hàm chi phí đối với các tham số.

- Tính toán độ dốc: Tại mỗi lần lặp, tính toán độ dốc của hàm chi phí đối với các tham số.
- Cập nhật trung bình chạy:

Cập nhật giá trị trung bình đang chạy bằng cách sử dụng phân rã theo cấp số nhân:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g^2 t \tag{1.8}$$

Ở đây γ là hệ số phân rã (thường gần bằng 1) xác định trọng số của đường trung bình chạy trước đó so với độ dốc hiện tại.

- Cập nhật thông số:

$$\Delta\theta_{t+1} = \Delta\theta_t - \frac{n\, g_t}{\sqrt{E[g^2]_t + \epsilon}} \tag{1.9}$$

Ở đây η là tốc độ học và ϵ là một hằng số nhỏ được thêm vào để ổn định số.

1.2.7 Adam(Adaptive Moment Estimation)

Adam (Adaptive Moment Estimation) là một kỹ thuật tối ưu hóa được sử dụng trong học máy và học sâu để tối ưu hóa việc đào tạo mạng lưới thần kinh.

Adam tích hợp các khái niệm về động lượng và RMSProp. Nó duy trì mức trung bình động của khoảnh khắc thứ nhất và thứ hai của gradient, tương ứng là giá trị trung bình và phương sai của gradient. Đường trung bình động của thời điểm ban đầu, có thể so sánh với số hạng động lượng trong các phương pháp tối ưu hóa khác, hỗ trợ trình tối ưu hóa tiếp tục tiến triển theo cùng một hướng ngay cả khi độ dốc nhỏ hơn. Trung bình động của thời điểm thứ hai, giống hệt với thuật ngữ RMSProp, hỗ trợ trình tối ưu hóa điều chỉnh tốc độ học cho từng tham số dựa trên phương sai của độ dốc.

Adam cũng bao gồm một giai đoạn hiệu chỉnh độ lệch để thay đổi các đường trung bình động vì chúng có độ lệch về 0 khi bắt đầu quá trình tối ưu hóa. Điều này giúp tăng hiệu suất của thuật toán tối ưu hóa trong giai đoạn đầu đào tạo.

Adam là một kỹ thuật tối ưu hóa phổ biến vì khả năng hội tụ nhanh chóng và quản lý độ dốc nhiễu hoặc thưa thớt. Ngoài ra, nó không yêu cầu cài đặt thủ công

các siêu tham số như suy giảm tốc độ học tập hoặc hệ số động lượng, giúp sử dụng dễ dàng hơn các kỹ thuật tối ưu hóa khác.

Cách hoạt động:

- Khởi tạo:

Khởi tạo các tham số (ước tính thời điểm ban đầu), (ước tính thời điểm thô thứ hai ban đầu).

Đặt t = 0 (bộ đếm lặp)

Chọn siêu tham số: α (tốc độ học), β_1 (tốc độ phân rã theo cấp số nhân cho ước tính khoảnh khắc đầu tiên), β_2 tốc độ phân rã theo cấp số nhân cho ước tính mô men thô thứ hai), ϵ (hằng số nhỏ để ổn định số).

- Tính toán độ dốc:

Tại mỗi lần lặp t, hãy tính độ dốc của hàm chi phí đối với các tham số.

- Cập nhật ước tính thời điểm đầu tiên:

Cập nhật ước tính thời điểm đầu tiên (momentum term):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1.10}$$

- Cập nhật ước tính khoảnh khắc thô thứ hai:

Cập nhật ước lượng mômen thô thứ hai:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(g_t)^2 \tag{1.11}$$

- Hiệu chỉnh sai lệch:

Sửa sai lệch trong ước tính thời điểm thứ nhất và thứ hai, có xu hướng sai lệch về 0 trong các lần lặp đầu tiên:

$$m_{t'} = \frac{m_t}{\beta_1^t} \tag{1.12}$$

$$v_{t'} = \frac{v_t}{\beta_2^t}$$

- Cập nhật tham số:

$$\Delta\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_{t'} + \epsilon}} m_{t'} \quad (1.13)$$

Ở đây θ thể hiện các thông số đang được tối ưu hóa.

- Lặp lại:

Tăng bộ đếm lần lặp t và lặp lại các bước 2-6 cho đến khi hội tụ hoặc số lần lặp xác định.

Nhìn chung, Adam là một phương pháp tối ưu hóa mạnh mẽ có thể giúp đẩy nhanh quá trình đào tạo mạng lưới thần kinh sâu và tăng hiệu suất của chúng.

1.3 Phân tích so sánh

Bảng 1.1: Phân tích so sánh các Optimizer

Optimizer	Ưu điểm	Nhược điểm
Gradient Descent	- Tính toán dễ dàng. - Dễ để thực hiện. - Dễ hiểu.	- Có thể mắc kẹt ở mức tối thiểu cục bộ. - Trọng số được thay đổi sau khi tính toán độ dốc trên toàn bộ tập dữ liệu. Vì vậy, nếu tập dữ liệu quá lớn thì có thể mất nhiều năm để hội tụ về mức tối thiểu. - Yêu cầu bộ nhớ lớn để tính toán độ dốc trên toàn bộ tập dữ liệu.
Stochastic Gradient Descent (SGD)	-Đánh giá ngay lập tức hiệu suất mô hình và tỷ lệ cải tiến. -Dễ hiểu và dễ thực hiện, đặc biệt dành cho người mới bắt đầu. - Cập nhật nhanh hơn tạo điều kiện phát hiện vấn đề nhanh hơn. -Tránh các cực tiểu cục bộ với quá trình cập nhật nhiễu.	- Tính toán chuyên sâu, đặc biệt với các tập dữ liệu lớn. - Cập nhật nhiễu có thể dẫn đến sự khác biệt giữa các epochs. -Quá trình học nhiễu có thể gây khó khăn cho việc cam kết sai số tối thiểu của mô hình.

	-Nhanh hơn và đòi hỏi ít sức mạnh tính toán hơn. -Thích hợp cho các tập dữ liệu lớn hơn.	
Momentum	- Giảm dao động trong quá trình tập luyện. - Hội tụ nhanh hơn cho các bài toán không điều kiện.	- Tăng độ phức tạp của thuật toán. - Chưa có hướng dẫn cụ thể cho việc lựa chọn cỡ mini-batch. -Thời gian đào tạo dài hơn do số lượng epochs có thể cao. -Khả năng vượt quá mức tối thiểu toàn cầu (global minimum).
Adagrad	- Tỷ lệ học tập thích ứng cho mỗi tham số. - Hiệu quả đối với dữ liệu thưa thớt.	- Việc tích lũy gradient bình phương trong mẫu số có thể khiến tốc độ học giảm xuống quá nhanh. - Có thể dừng việc học quá sớm.
Adadelta	- Có thể điều chỉnh tốc độ học tập linh hoạt hơn Adagrad. - Không có siêu tham số tốc độ học tập.	- Việc thích ứng tốc độ học có thể quá tích cực, dẫn đến tốc độ hội tụ chậm.
RMSProp	- Tốc độ học thích ứng trên mỗi tham số giúp hạn chế sự tích lũy độ dốc. - Hiệu quả đối với các mục tiêu không cố định.	- Có thể có tốc độ hội tụ chậm trong một số trường hợp.
Adam	-Dễ để thực hiện. -Tính toán hiệu quả. -Yêu cầu bộ nhớ nhỏ. - Bất biến đối với sự thay đổi tỷ lệ đường chéo của độ dốc.	- Yêu cầu điều chỉnh cẩn thận các siêu tham số.

	-Rất phù hợp với các bài toán có lượng dữ liệu và/hoặc tham số lớn. -Thích hợp cho các mục tiêu không cố định. -Siêu tham số có thể diễn giải trực quan và thường cần ít điều chỉnh. -Tốc độ và động lực học thích ứng cho từng tham số -Tốc độ học không giảm như AdaGrad	

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

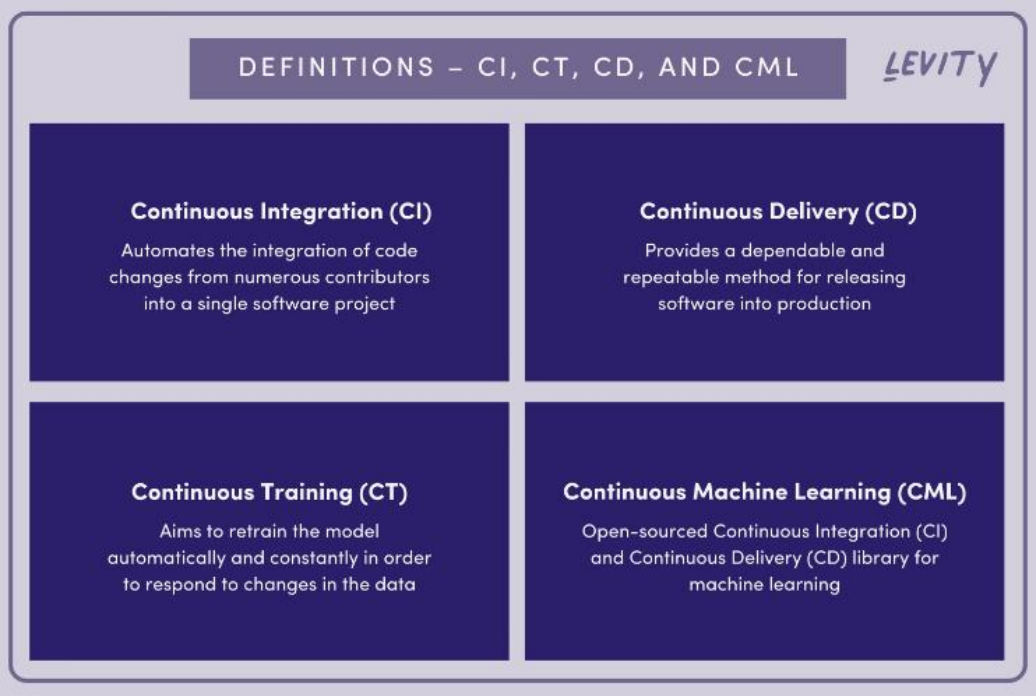
2.1 Continual Learning

2.1.1 Khái niệm

Học máy là một loại Trí tuệ nhân tạo (AI) cho phép máy tính có khả năng học hỏi mà không cần được con người đào tạo rõ ràng để làm việc đó. Ở cấp độ cơ bản, Machine Learning sử dụng các thuật toán để cung cấp cho máy tính khả năng nghiên cứu dữ liệu, phát hiện các mẫu và tạo kết quả dự đoán. Tuy nhiên, khó khăn là mô hình Machine Learning tiêu chuẩn dự đoán rằng dữ liệu sẽ có thể so sánh được với dữ liệu mà nó đã được đào tạo, điều này không nhất thiết phải như vậy.

Continuous Machine Learning (CML) giải quyết vấn đề này bằng cách giám sát và đào tạo lại các mô hình với dữ liệu hiện tại. Mục đích của CML là mô phỏng khả năng thu thập và tinh chỉnh kiến thức liên tục của con người. Trong bài viết này, chúng ta sẽ đi sâu vào chi tiết về chính xác CML là gì, những trở ngại liên quan đến nó và tại sao nó lại quan trọng đối với sự phát triển của AI.

Trước khi đi vào chi tiết về Continuous Machine Learning, điều quan trọng trước tiên là phải xác định các công nghệ DevOps mà nó dựa vào – CI, CT và CD.



Hình 2.1: Định nghĩa CI, CT, CD và CML

(Nguồn: (Levity, 2022))

- **Continuous Integration (CI)**

Hoạt động tự động hóa việc tích hợp các thay đổi mã từ một số người đóng góp vào một dự án phần mềm duy nhất được gọi là Tích hợp liên tục (CI). Đó là phương pháp thực hành tốt nhất của DevOps cho phép các nhà phát triển thường xuyên hợp nhất các thay đổi mã vào một trung tâm duy nhất nơi các bản dựng và thử nghiệm sau đó được thực hiện. Trước khi kết hợp mã mới, các công cụ tự động sẽ được kiểm tra để xác nhận tính chính xác của chúng.

Cách tiếp cận CI phụ thuộc đáng kể vào hệ thống kiểm soát phiên bản mã nguồn. Ngoài ra, các kiểm tra sâu hơn, chẳng hạn như kiểm tra chất lượng mã tự động, các công cụ đánh giá kiểu cú pháp và các công cụ khác, được đưa vào hệ thống kiểm soát phiên bản.

- **Continuous Training (CT)**

Như đã nêu trước đây, các mô hình Machine Learning (ML) giả định rằng dữ liệu sẽ luôn có thể so sánh được với dữ liệu mà nó đã được đào tạo. Tuy nhiên, không phải lúc nào cũng như vậy. Cụ thể, phần lớn các mô hình hoạt động trong các

tình huống mà dữ liệu thay đổi thường xuyên và có khả năng xảy ra hiện tượng "trôi dạt khái niệm", điều này có thể có ảnh hưởng bất lợi đến độ chính xác và độ tin cậy của các dự đoán của mô hình. Để tránh tình trạng "trôi dạt khái niệm" trong quá trình phát triển , các mô hình cần được kiểm tra và đào tạo lại khi dữ liệu quá sai.

Đây là nơi xuất hiện khái niệm Đào tạo liên tục. CT là một thành phần của mô hình thực hành MLOps. Nó tìm cách đào tạo lại mô hình một cách tự động và liên tục để thích ứng với những thay đổi trong dữ liệu. Cách làm này tránh cho mô hình trở nên không đáng tin cậy và không chính xác.

- **Continuous Delivery (CD)**

Mục đích của Phân phối liên tục là tạo ra một cơ chế đáng tin cậy và có thể lặp lại để triển khai phần mềm vào sản xuất. Phân phối liên tục cho Machine Learning (CD4ML) mở rộng phương pháp này bằng cách cho phép một nhóm đa chức năng xây dựng các ứng dụng Machine Learning dựa trên mã, dữ liệu và mô hình cải thiện theo từng bước nhỏ, an toàn có thể được sao chép và xuất bản một cách đáng tin cậy bất cứ lúc nào.

- **Continuous Machine Learning (CML)**

Bây giờ chúng ta đã hiểu rõ về các nguyên tắc trên, chúng ta có thể xem xét kỹ hơn một chút về Học máy liên tục.

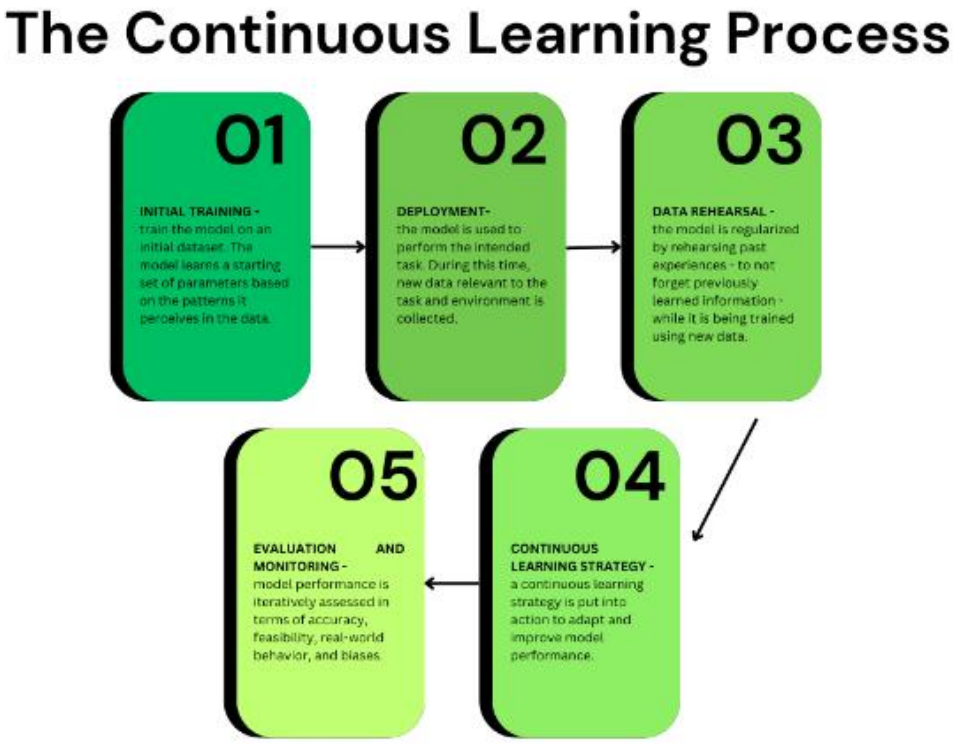
CML, viết tắt của Học máy liên tục, là thư viện Tích hợp liên tục (CI) và Phân phối liên tục (CD) mã nguồn mở dành cho Học máy. Nói chung, nó có thể được sử dụng để tự động hóa các yếu tố trong quy trình Machine Learning của bạn, chẳng hạn như đào tạo và đánh giá mô hình, so sánh các thử nghiệm ML trong lịch sử dự án của bạn và theo dõi các thay đổi trong bộ dữ liệu. Nó hoạt động trên nền tảng của hệ sinh thái MLOps, cho phép bạn sử dụng các công cụ DevOps ưa thích của mình trong các dự án Machine Learning.

Nếu bạn đã quen với hệ thống đề xuất của Netflix, hệ thống này có tính năng “Tiếp theo” phát các chương trình tương tự như những chương trình bạn vừa xem thì bạn đã thấy mô hình CML hoạt động. Để theo kịp nguồn cung cấp chương trình mới không ngừng nghỉ, cũng như sự thay đổi sở thích và thói quen của người đăng

ký Netflix, dữ liệu đến phải được nhập thường xuyên. Và mô hình phải cập nhật liên tục để có khả năng đề xuất các chương trình hoặc bộ phim phù hợp.

2.1.2 Quá trình Continual Learning

Học liên tục thể hiện sự tiến bộ so với các phương pháp học máy thông thường, kết hợp các nguyên tắc lập mô hình quen thuộc như tiền xử lý, lựa chọn mô hình, điều chỉnh siêu tham số, đào tạo, triển khai và giám sát. Để nâng cao khả năng thích ứng với việc phát triển dữ liệu, hai bước bổ sung trở nên cần thiết trong quá trình học tập liên tục: diễn tập dữ liệu và xây dựng chiến lược học tập liên tục. Các bước này được thiết kế để tối ưu hóa việc đồng hóa các luồng dữ liệu mới của mô hình, phù hợp với các yêu cầu và bối cảnh cụ thể của nhiệm vụ dữ liệu.



Hình 2.2: Sơ đồ quy trình học tập liên tục đến học máy

2.1.3 Các loại Continual Learning

2.1.3.1 Incremental Learning

- Kiến trúc động:

Mô tả: Sử dụng các mô hình có kiến trúc có thể thích ứng và mở rộng để kết hợp kiến thức mới mà không quên nhiệm vụ cũ.

Ứng dụng: Mạng có các nút động hoặc cấu trúc có thể thích ứng có thể phát triển tăng dần.

- Progressive Neural Networks (PNN):

Mô tả: Cho phép bổ sung thêm các đơn vị mới vào mạng để học các nhiệm vụ mới mà không ảnh hưởng đến kiến thức hiện có.

Ứng dụng: Mở rộng dần dần mạng lưới thần kinh theo thời gian.

- Tinh chỉnh:

Mô tả: Đào tạo lại mô hình về các nhiệm vụ mới với tốc độ học tập nhỏ hơn để thích ứng với những thay đổi gia tăng.

Ứng dụng: Cập nhật các tham số của mô hình để phù hợp với các nhiệm vụ mới trong khi vẫn giữ lại kiến thức từ các nhiệm vụ trước đó.

2.1.3.2 Transfer Learning

- Khai thác tính năng:

Mô tả: Sử dụng các mô hình được đào tạo trước cho nhiệm vụ nguồn và chuyển giao kiến thức bằng cách trích xuất các tính năng và chỉ đào tạo các lớp cuối cùng về nhiệm vụ đích.

Ứng dụng: Áp dụng kiến thức thu được từ lĩnh vực này sang lĩnh vực liên quan khác.

- Thích ứng tên miền:

Mô tả: Điều chỉnh mô hình được đào tạo trước từ một miền để hoạt động hiệu quả trong một miền khác nhưng có liên quan.

Ứng dụng: Chuyển giao kiến thức giữa các lĩnh vực với những thay đổi nhỏ.

- Multi-Task Learning (Học đa nhiệm vụ):

Mô tả: Đồng thời đào tạo một mô hình về nhiều nhiệm vụ, tận dụng các biểu diễn được chia sẻ.

Ứng dụng: Tìm hiểu một không gian đặc trưng chung có lợi cho nhiều nhiệm vụ.

2.1.3.3 Lifelong Learning

- Mạng tăng cường bộ nhớ:

Mô tả: Tích hợp bộ nhớ ngoài để lưu trữ và truy xuất thông tin đã học theo thời gian.

Ứng dụng: Lưu giữ kiến thức quan trọng từ các nhiệm vụ trước đó vào bộ nhớ ngoài.

- Kiến trúc nhận biết nhiệm vụ:

Mô tả: Thiết kế các mô hình nhận thức được các nhiệm vụ mà chúng hiện đang thực hiện, cho phép điều chỉnh theo từng nhiệm vụ cụ thể.

Ứng dụng: Tự động điều chỉnh kiến trúc dựa trên nhiệm vụ hiện tại.

- Meta-Learning (Siêu học tập):

Mô tả: Huấn luyện mô hình để thích ứng nhanh với các nhiệm vụ mới dựa trên kinh nghiệm thu được từ các nhiệm vụ trước đó.

Ứng dụng: Học hỏi nhanh và thích ứng với các nhiệm vụ mới với lượng dữ liệu tối thiểu.

2.1.3.4 Experience Replay Methods

- Bộ đệm phát lại:

Mô tả: Lưu trữ và lấy mẫu ngẫu nhiên các trải nghiệm từ bộ đệm dữ liệu trong quá khứ trong quá trình đào tạo.

Ứng dụng: Giảm thiểu tình trạng quên lãng nghiêm trọng bằng cách định kỳ xem lại và rèn luyện những kinh nghiệm trong quá khứ.

- Hợp nhất trọng lượng đàn hồi (EWC):

Mô tả: Chuẩn hóa các tham số của mô hình để bảo vệ các trọng số quan trọng đã học được trong các tác vụ trước đó.

Ứng dụng: Ngăn chặn việc quên bằng cách gán hình phạt cao hơn cho các tham số quan trọng.

2.1.3.5 Regularization Techniques:

- EWC (Elastic Weight Consolidation):

Mô tả: Đưa ra số hạng phạt trong hàm mất mát để bảo toàn các trọng số quan trọng.

Ứng dụng: Bảo vệ các thông số quan trọng trong quá trình đào tạo các nhiệm vụ mới.

- Chính quy hóa L2:

Mô tả: Thêm một số hạng phạt vào hàm mất mát dựa trên độ lớn bình phương của các trọng số.

Ứng dụng: Kiểm soát quá mức và duy trì mô hình ổn định hơn.

- Dropout:

Mô tả: Vô hiệu hóa ngẫu nhiên các tế bào thần kinh trong quá trình huấn luyện để ngăn chặn sự đồng thích ứng của các đơn vị ẩn.

Ứng dụng: Cải thiện khả năng khái quát hóa bằng cách giảm sự phụ thuộc vào các nơ-ron cụ thể.

Trong Continual Learning, việc kết hợp các phương pháp này một cách thận trọng thường là cần thiết để giải quyết những thách thức đặc biệt liên quan đến việc học theo thời gian mà không quên kiến thức quá khứ. Các nhà nghiên cứu tiếp tục khám phá các kỹ thuật mới để nâng cao khả năng của các mô hình học tập liên tục.

2.1.4 Continual Learning hoạt động như thế nào trong học máy

Continual Learning, còn được gọi là lifelong learning hoặc incremental learning, là một lĩnh vực học máy tập trung vào khả năng học hỏi và thích ứng của hệ thống theo thời gian khi có dữ liệu mới. Mục tiêu là cho phép một mô hình học hỏi từ một luồng dữ liệu mà không quên kiến thức mà nó đã thu được từ những trải nghiệm trước đó. Dưới đây là phần giải thích từng bước về cách hoạt động của học tập liên tục trong học máy, cùng với một số thuật toán và phương trình

- Định nghĩa:

Xác định các nhiệm vụ mà mô hình cần học theo thời gian. Nhiệm vụ có thể là phân phối dữ liệu khác nhau, vấn đề phân loại hoặc bất kỳ mục tiêu học tập nào khác.

- Khởi tạo:

Khởi tạo mô hình với một số tham số ban đầu. Điều này thường được thực hiện bằng cách sử dụng đào tạo tiêu chuẩn trên tập dữ liệu ban đầu cho nhiệm vụ đầu tiên.

- Đào tạo nhiệm vụ:

Huấn luyện mô hình về nhiệm vụ hiện tại bằng cách sử dụng dữ liệu có sẵn. Các thuật toán học có giám sát tiêu chuẩn như giảm độ dốc có thể được sử dụng ở đây.

$$Loss_{task}(\theta) = \sum_{i=1}^{N} Loss(f(xi \quad ;\theta), yi \quad) \tag{2.1}$$

Trong đó θ đại diện cho các tham số của mô hình, N là số lượng mẫu huấn luyện, xi là ví dụ đầu vào, yi là nhãn tương ứng và f là hàm dự đoán của mô hình.

- Cập nhật thông số:

Cập nhật các tham số mô hình để giảm thiểu tổn thất theo nhiệm vụ cụ thể.

$$\theta_{new} \quad = \theta - \eta \nabla_{\theta} \quad \text{Loss}_{\text{task}} \quad (\theta) \tag{2.2}$$

Trong đó η là tốc độ học và ∇_{θ} biểu thị độ dốc đối với các tham số mô hình.

- Kỹ thuật chính quy hóa:

Áp dụng các kỹ thuật chính quy hóa để ngăn chặn sự quên lãng thảm khốc. Các phương pháp chính quy hóa, chẳng hạn như hợp nhất trọng số đàn hồi (EWC) hoặc trí thông minh khớp thần kinh (SI), có thể được sử dụng để xử phạt các thay đổi đối với các tham số quan trọng đã học trong các nhiệm vụ trước đó.

- Phát lại bộ nhớ

Lưu trữ các ví dụ đại diện từ các tác vụ trước đó vào bộ nhớ đệm và phát lại chúng trong quá trình đào tạo về các tác vụ mới. Điều này giúp lưu giữ thông tin về các nhiệm vụ trong quá khứ và giảm thiểu tình trạng quên.

$$Loss_{task}(\theta) = \sum_{j=1}^{M} Loss(f(xi \quad ;\theta), yi \quad) \tag{2.3}$$

Trong đó M là số mẫu trong bộ nhớ đệm.

- Chuyển đổi nhiệm vụ

Lặp lại các bước 3-6 cho mỗi nhiệm vụ trong trình tự. Khi các nhiệm vụ mới được giới thiệu, mô hình sẽ điều chỉnh các tham số của nó để thực hiện tốt cả nhiệm vụ mới và nhiệm vụ đã học trước đó.

- Đánh giá

Đánh giá định kỳ hiệu suất của mô hình trên tất cả các nhiệm vụ để đảm bảo rằng nó duy trì mức độ chính xác tốt đối với các nhiệm vụ đã học trước đó trong khi vẫn học các nhiệm vụ mới.

2.1.5 Ưu điểm và hạn chế của quá trình Continual Learning

- Ưu điểm của Continual Learning

Học liên tục chứng tỏ có giá trị trong nhiều dự án dữ liệu, bao gồm phân tích mô tả, chẩn đoán, dự đoán và phân tích theo quy định. Tầm quan trọng của nó được đặc biệt nhấn mạnh trong các tình huống có dữ liệu thay đổi nhanh chóng. So với các phương pháp học máy thông thường, các ưu điểm bao gồm:

- Tăng cường khái quát hóa:

Học tập liên tục giúp mô hình thể hiện độ tin cậy và độ chính xác cao hơn khi đối mặt với dữ liệu mới.

Ưu điểm: Cải thiện khả năng khái quát hóa trên các tình huống đa dạng và đang phát triển.

- Lưu giữ thông tin:

Thông qua việc áp dụng các chiến lược học tập liên tục, mô hình sẽ tính đến kiến thức trước đó thu được trong các lần lặp trước.

Ưu điểm: Mô hình có thể tích lũy thông tin theo thời gian, giảm thiểu nguy cơ quên các mẫu đã học trước đó.

- Khả năng thích ứng:

Các mô hình sử dụng học tập liên tục thể hiện khả năng thích ứng với kiến thức mới, chẳng hạn như sự trôi dạt khái niệm và các xu hướng mới nổi.

Ưu điểm: Khả năng dự đoán cao hơn trong thời gian dài, đảm bảo tính phù hợp khi đối mặt với môi trường dữ liệu động.

- Hạn chế của Continual Learning

Mặc dù việc học liên tục mang lại khả năng thích ứng hiệu quả với dữ liệu mới, nhưng các thách thức về mô hình và chi phí liên quan phải được xem xét cẩn thận. Độ phức tạp tính toán tăng lên có thể dẫn đến chi phí kinh tế cao hơn và các vấn đề như quản lý mô hình và trôi dạt dữ liệu đặt ra những thách thức bổ sung trong việc thực hiện các phương pháp học tập liên tục.

- Cân nhắc về chi phí và nhược điểm của mô hình hóa:

Học tập liên tục, mặc dù có hiệu quả trong việc thích ứng với dữ liệu mới, nhưng lại đưa ra những cân nhắc nhất định về chi phí và những hạn chế trong mô hình hóa. Từ góc độ chi phí, các phương pháp học liên tục có xu hướng phức tạp hơn về mặt tính toán so với các phương pháp truyền thống. Sự phức tạp ngày càng tăng này phát sinh từ việc mô hình liên tục thích ứng với dữ liệu mới, đòi hỏi đầu tư bổ sung vào dữ liệu, nguồn nhân lực và cơ sở hạ tầng máy tính. Các chi phí kinh tế liên quan đến những nhu cầu này có thể là một sự cân nhắc đáng kể đối với các tổ chức.

- Về mặt mô hình hóa, việc học tập liên tục có những hạn chế:

Đầu tiên, vấn đề quản lý mô hình nảy sinh. Với mỗi lần cập nhật dựa trên dữ liệu mới, một mô hình mới sẽ được hình thành, có khả năng dẫn đến một số lượng lớn các mô hình. Sự gia tăng nhanh chóng này làm phức tạp thêm việc xác định các mô hình hoạt động tốt nhất và đặt ra những thách thức về mặt quản trị và đánh giá mô hình.

Thứ hai, thách thức về sự trôi dạt dữ liệu gây rủi ro cho tính hiệu quả của các phương pháp học tập liên tục. Học tập liên tục phụ thuộc vào việc xử lý khối lượng lớn dữ liệu mới và những thay đổi đột ngột trong phân phối tính năng, được gọi là trôi dạt dữ liệu, có thể làm suy yếu khả năng dự đoán của mô hình. Việc giải quyết

thành công tình trạng trôi dạt dữ liệu trở thành một thách thức quan trọng, đòi hỏi phải thực hiện các chiến lược để duy trì độ chính xác của mô hình khi đối mặt với môi trường dữ liệu động.

2.1.6 Ứng dụng của Continual Learning

Thị giác máy tính: Học liên tục được sử dụng để đào tạo các thuật toán trong các nhiệm vụ nhận dạng và phân loại hình ảnh, chẳng hạn như nhận dạng khuôn mặt và phát hiện đối tượng.

An ninh mạng: Các phương pháp học hỏi liên tục được sử dụng để giám sát liên tục các hệ thống bảo mật CNTT và phát hiện các mối đe dọa như nỗ lực lừa đảo, xâm nhập mạng và thư rác, đảm bảo trạng thái bảo mật chủ động.

Chăm sóc sức khỏe: Học tập liên tục được áp dụng trong chăm sóc sức khỏe để cải thiện quy trình chẩn đoán, đặc biệt là trong các lĩnh vực như ung thư và X quang, nơi tính chất biến động của bệnh đòi hỏi phải học hỏi và thích nghi liên tục.

Robotics: Các kỹ thuật học tập liên tục được sử dụng để nâng cao khả năng thích ứng và hiệu suất của robot. Bằng cách liên tục học hỏi từ những trải nghiệm mới và tối ưu hóa hành động, robot có thể hoạt động hiệu quả hơn trong những môi trường luôn thay đổi.

Xử lý ngôn ngữ tự nhiên: Học tập liên tục được sử dụng trong các ứng dụng NLP như chatbot và trợ lý ảo, cho phép họ cải thiện khả năng hiểu ngôn ngữ và tạo phản hồi theo thời gian thông qua các tương tác liên tục.

2.2 Test Production

2.2.1 Giới thiệu

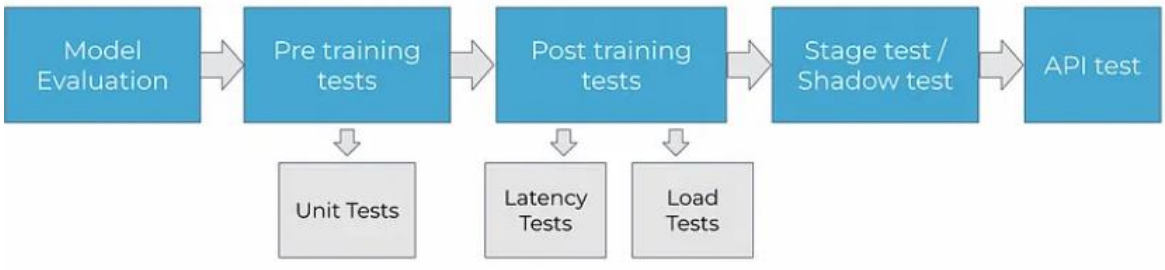
Các bài kiểm tra học máy có thể được phân thành hai loại: bài kiểm tra phần mềm tiêu chuẩn và bài kiểm tra học máy (ML). Kiểm tra phần mềm đánh giá logic bằng văn bản, trong khi kiểm tra ML đánh giá logic đã học.

Các bài kiểm tra ML có thể được chia thành hai thành phần riêng biệt: kiểm tra và đánh giá. Chúng tôi đã làm quen với đánh giá ML, bao gồm việc đào tạo một mô hình và đánh giá hiệu suất của nó trên một bộ xác thực chưa từng thấy trước đây.

Đánh giá này được thực hiện bằng cách sử dụng các số liệu như độ chính xác và Vùng dưới đường cong của đặc tính hoạt động của máy thu (AUC ROC), cũng như đồ họa như đường cong thu hồi chính xác

Ngược lại, thử nghiệm ML đòi hỏi phải đánh giá hành vi của mô hình. Các bài kiểm tra đào tạo trước, có thể được thực hiện mà không cần dạy các tham số, xác minh tính chính xác của logic bằng văn bản của chúng tôi. Chẳng hạn, xác suất phân loại có bị giới hạn trong phạm vi từ 0 đến 1 không? Đánh giá sau đào tạo đánh giá mức độ phù hợp của lý luận thu được với kết quả dự kiến. Trên tập dữ liệu Titanic, thật hợp lý khi dự đoán rằng phụ nữ sẽ có khả năng sống sót cao hơn so với nam giới.

2.2.2 Quá trình thử nghiệm



Hình 2.3: Luồng thử nghiệm học máy

2.2.2.1 Đánh giá mô hình

Model evaluation is stage 0 of model testing and is only limited to the functionality of the model. Below is the metric for each type of model to evaluate its quality. For imbalanced datasets, F1 scores, and AUC scores are best for classification and for outlier-heavy data, MAE is better for regression. For some models like the ensemble-decision tree-based supervised model (XGBoost, RF). SHAP can be used to understand what is the logic of prediction as such a black box model at the overall view and LIME for specific cases inspection

2.2.2.2 Pre-Training Test/ Unit Testing

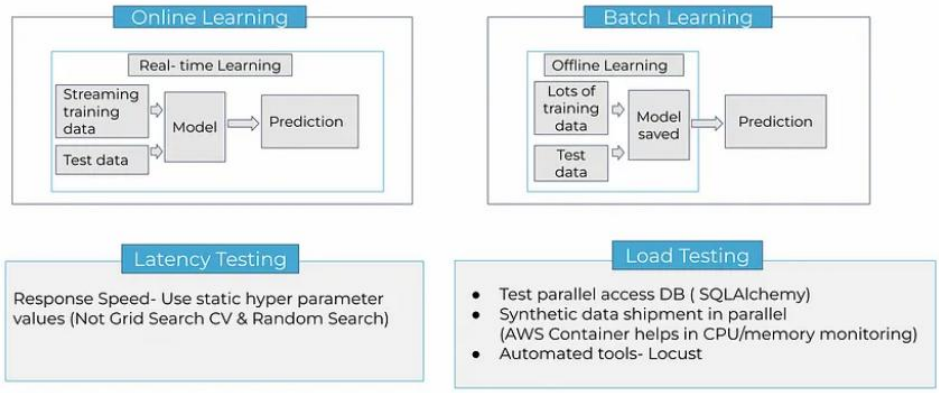
Trước quá trình đào tạo theo cả phương pháp học trực tuyến và theo đợt, việc kiểm tra đơn vị được tiến hành để đảm bảo rằng phần mềm/mô hình đáp ứng các tiêu chí cần thiết, chẳng hạn như định dạng dữ liệu chính xác và đủ dữ liệu.

2.2.2.3 Post Training Test- Batch Vs Online Learning

Bài kiểm tra sau đào tạo được thực hiện sau khi hoàn thành khóa đào tạo về học theo đợt, trong khi nó được thực hiện trong quá trình đào tạo học trực tuyến.

Kiểm tra độ trễ: Kiểm tra xem dự đoán có được tạo ra trong vòng một phần giây hay không để mô hình có thể mở rộng và quản lý lưu lượng. Nếu mất hơn một phút, phần lớn thiết kế mô hình cần phải được thay đổi. Bài kiểm tra này đặc biệt quan trọng nếu nó là một kỹ thuật học máy trực tuyến. Một cách để giải quyết vấn đề này trong học trực tuyến là xác định siêu tham số là giá trị tĩnh và không sử dụng cv tìm kiếm dạng lưới để điều chỉnh tham số mỗi khi có dữ liệu mới vì nó có thể làm tăng độ trễ. Do đó, giá trị tĩnh của siêu tham số có thể được phát hiện bằng cách chạy tập dữ liệu được lấy mẫu/tập hợp con và huấn luyện mô hình bằng cách sử dụng các phương pháp như cv tìm kiếm ngẫu nhiên để thu được các tham số tối ưu dựa trên dữ liệu xác thực đa số

Kiểm tra tải: Kiểm tra xem mô hình có thể xử lý bao nhiêu dữ liệu kiểm tra cùng một lúc. Điều này rất quan trọng cho cả việc học theo đợt và học trực tuyến. Một cách tiếp cận là sử dụng SQLalchemy để xác minh xem tất cả DB (cơ sở dữ liệu) có được truy cập song song hay không, điều này sẽ cho phép xử lý lượng dữ liệu khổng lồ. Ngoài ra, bộ chứa AWS còn giúp giám sát CPU/bộ nhớ. Locust là một công cụ để tự động hóa việc kiểm tra này



Hình 2.4: Post training test

2.2.2.4 A/B testing

Với một số mô hình ML/AI nhất định, theo thời gian, các tính năng của dữ liệu sẽ thay đổi, do đó mô hình được đào tạo trên dữ liệu cũ không thể hoạt động tốt với dữ liệu mới. Điều này được gọi là sự trôi dạt dữ liệu. Tương tự, hiện tượng trôi dạt ý tưởng xảy ra khi các giả định do mô hình học máy tạo ra không còn đúng trong dữ liệu trong thế giới thực mà nó đáp ứng sau khi triển khai.

Vì vậy, việc đào tạo lại là điều cần thiết sau khi triển khai ML. Thử nghiệm A/B giúp xác định xem phiên bản mới của mô hình có nên thay thế phiên bản cũ hay không.

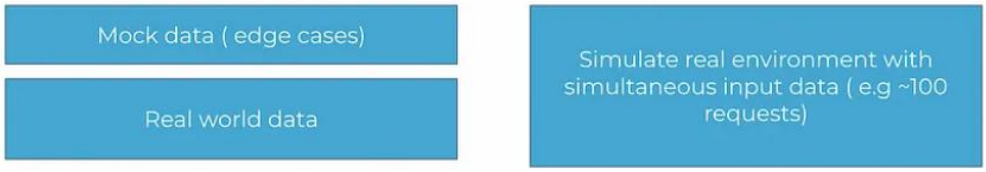
Trong thử nghiệm A/B (được AWS Sagemaker hỗ trợ tự động), 80% lưu lượng truy cập được cung cấp cho mô hình ml hiện tại/cũ trong khi 20% lưu lượng truy cập (đối thủ) được cung cấp cho mô hình mới. Dựa trên (các) số liệu cơ sở, mô hình mới có thể thay thế mô hình cũ nếu mô hình mới hoạt động tốt hơn.

2.2.2.5 Stage test/ Shadow test

Thử nghiệm Giai đoạn là một trong những thử nghiệm cuối cùng để xác định xem liệu mô hình có mang lại kết quả như mong đợi hay không. Thử nghiệm này diễn ra sau quá trình dockerization và AWS containerization để triển khai tự động trong các quy trình như Bitbucket và GitLab. Phần này chứa nhiều dữ liệu thử nghiệm khác nhau (bao gồm các kịch bản rộng lớn) thể hiện các tính năng dữ liệu trong thế giới thực và được phân phối dưới dạng đầu vào cho mô hình ở môi trường giai đoạn.

Kiểm tra bóng (Kỹ thuật an toàn để kiểm tra độ tỉnh táo khi triển khai các mô hình ML/AI lớn):

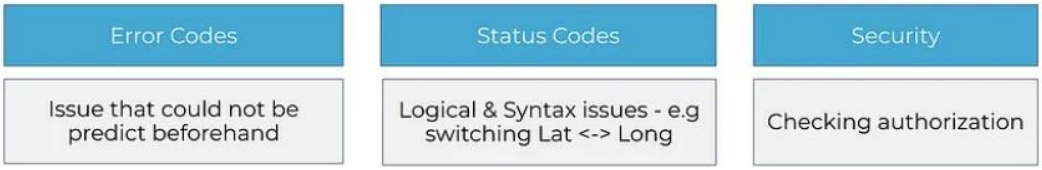
- Triển khai mô hình song song với mô hình hiện tại (nếu đã có).
- Đối với mỗi yêu cầu, hãy định tuyến yêu cầu đó đến cả hai mô hình để tạo dự báo nhưng chỉ phân phối dự báo hiện tại cho người dùng
- Sử dụng dự đoán trên mô hình mới để đánh giá/phân tích



Hình 2.5: Stage Test

2.2.2.6 API Testing

Đây là thử nghiệm cuối cùng, nơi chúng tôi xác minh cách người dùng thực sự sẽ xem câu trả lời từ mô hình. Do đó, nó nắm bắt tất cả các tình huống khác nhau trong đầu vào của người dùng có thể tạo ra sự cố khi phản hồi. Mỗi lỗi được mã hóa dưới dạng một Id duy nhất (dưới dạng status_code, mã lỗi). Và khía cạnh bảo mật của đầu vào và đầu ra được kiểm tra cho từng người tiêu dùng riêng biệt.



Hình 2.6: API Testing

2.2.3 Các loại testing production

Có bốn loại thử nghiệm chính được sử dụng ở các giai đoạn khác nhau trong chu kỳ phát triển:

- Unit tests: kiểm tra trên các thành phần riêng biệt mà mỗi thành phần có một trách nhiệm duy nhất (ví dụ: chức năng lọc danh sách).

- Integration tests: kiểm tra chức năng tích hợp của các thành phần riêng biệt (ví dụ: xử lý dữ liệu).

- System tests: kiểm tra kiến trúc của hệ thống về các kết quả đầu ra dự kiến dựa trên các đầu vào (ví dụ: đào tạo, suy luận, v.v.).

- Acceptance tests: kiểm tra để xác minh rằng các tiêu chí đã được thỏa mãn, thường được gọi là Kiểm tra chấp nhận của người dùng (UAT).

- Regression tests: kiểm tra dựa trên các lỗi mà chúng tôi đã gặp phải trước đây để đảm bảo các sửa đổi mới không lặp lại chúng.

Mặc dù các hệ thống ML có bản chất xác suất nhưng chúng được tạo thành từ nhiều thành phần có thể dự đoán được và có thể được xác minh theo cách tương tự như các hệ thống phần mềm thông thường. Sự tương phản giữa các hệ thống ML thử nghiệm xuất hiện khi chúng tôi chuyển từ thử nghiệm mã sang thử nghiệm dữ liệu và mô hình.

TÀI LIỆU THAM KHẢO

Tiếng Anh

Musstafa. (2021). *Optimizers in Deep Learning.* Retrieved December 22, 2023, from Medium: https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0

Hanna Kleinings. (2022). *What is Continuous Machine Learning?.* Retrieved December 22, 2023, from Levity: https://levity.ai/blog/what-is-continuous-machine-learning

Sanket Doshi. (2019). *Various Optimization Algorithms For Training Neural Network.* Retrieved December 22, 2023, from towardsdatascience: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6