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

ĐỒ ÁN CUỐI KÌ

NHẬP MÔN HỌC MÁY

Người hướng dẫn: **GV. LÊ ANH CƯỜNG**
Người thực hiện: **TRẦN THỊ VẸN – 52100674**
Lớp **: 21050301**
Khoá **: 25**

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

ĐỒ ÁN CUỐI KÌ

NHẬP MÔN HỌC MÁY

Người hướng dẫn: **GV. LÊ ANH CƯỜNG**
Người thực hiện: **TRẦN THỊ VẸN – 52100674**
Lớp **: 21050301**
Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Trong suốt quá trình học tập và rèn luyện, chúng em đã nhận được rất nhiều sự giúp đỡ tận tình, sự quan tâm, chăm sóc của GV.Lê Anh Cường. Ngoài ra, chúng em còn được GV truyền đạt những kiến thức, phương pháp mới về toán hay ho và thú vị, thầy cô còn giúp sinh viên có được nhiều niềm vui trong việc học và cảm thấy thoải mái, … Chúng em xin chân thành cảm ơn các thầy cô rất nhiều trong suốt quá trình học tập này!

Bởi lượng kiến thức của chúng em còn hạn hẹp và gặp nhiều vấn đề trong quá trình học nên báo cáo này sẽ còn nhiều thiếu sót và cần được học hỏi thêm. Chúng em rất mong em sẽ nhận được sự góp ý của quý thầy cô về bài báo cáo này để chúng em rút kinh nghiệm trong những môn học sắp tới. Cuối cùng, chúng em xin chân thành cảm ơn quý thầy cô.

TP Hồ Chí Minh, ngày 10 tháng 12 năm 2023

Sinh viên:

Trần Thị Vẹn – 52100674

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

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

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

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

TP. Hồ Chí Minh, ngày 10 tháng 12 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Trần Thị Vẹn

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

Để giải quyết bài toán phân lớp hình ảnh trong huấn luyện mô hình học sâu, việc tinh chỉnh tốc độ học luôn là một trong những ưu tiên hàng đầu. Do đó, việc lựa chọn thuật toán tối ưu thích hợp cho mô hình là cần thiết. Trong đồ án cuối kì này, tôi đánh giá các thuật toán tối ưu phổ biến và tác động của chúng đến quá trình huấn luyện của mô hình học sâu.

MỤC LỤC

CHƯƠNG 1: CÁC PHƯƠNG PHÁP OPTIMIZER

Trước khi đi sâu vào vấn đề thì chúng ta cần hiểu thế nào là thuật toán tối ưu (optimizers).Về cơ bản, thuật toán tối ưu là cơ sở để xây dựng mô hình neural network với mục đích "học " được các features (hay pattern) của dữ liệu đầu vào, từ đó có thể tìm 1 cặp weights và bias phù hợp để tối ưu hóa model.

Trong học sâu, các thuật toán tối ưu hóa là những thuật toán điều chỉnh các tham số của mô hình trong quá trình huấn luyện để giảm thiểu một hàm mất mát. Chúng cho phép các mạng nơ-ron học từ dữ liệu bằng cách cập nhật trọng số và độ lệch theo từng vòng lặp. Các thuật toán tối ưu phổ biến bao gồm Gradient Descent Stochastic (SGD), Adam và RMSprop.

Những thuật toán tối ưu hóa này đóng vai trò quan trọng trong việc đào tạo mạng nơ-ron, giúp mô hình học hiệu quả và nhanh chóng từ dữ liệu. Trong quá trình huấn luyện mô hình học máy, việc sử dụng một phương pháp tối ưu hóa (optimizer) phù hợp có thể ảnh hưởng lớn đến hiệu suất và tốc độ học của mô hình. Dưới đây là một số phương pháp tối ưu hóa phổ biến và so sánh giữa chúng:

1.1 Batch, Mini Batch & Stochastic Gradient Descent

Gradient Descent (GD) là một trong những thuật toán phổ biến nhất khi tối ưu hóa mạng nơron. GD tối thiểu hóa hàm mất mát (loss function) J(θ) trong đó θ là tập hợp các trọng số của mô hình cần tối ưu. Quy tắc cập nhật của GD ở dạng tổng quát như sau:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta_t)$$

Trong đó:

- $\nabla_\theta J(\theta_t)$ là gradient của hàm mất mát tại θ ở bước t . η là một số dương được gọi là tốc độ học (learning rate).
- tốc độ học η xác định kích thước của các bước di chuyển đến giá trị cực tiểu (hoặc cực tiểu địa phương).

Có một số biến thể khác nhau của GD tùy thuộc vào số lượng dữ liệu được sử dụng để tính gradient của hàm mất mát. Thuật toán **Batch Gradient Descent** (Batch GD)

tính gradient của hàm mất mát tại θ trên toàn bộ tập dữ liệu. Tất cả các điểm dữ liệu đều được sử dụng để tính gradient trước khi cập nhật bộ trọng số θ . Hạn chế của Batch GD là khi tập dữ liệu lớn, việc tính gradient sẽ tốn nhiều thời gian và chi phí tính toán. Để khắc phục hạn chế này, thuật toán Stochastic Gradient Descent (SGD) thực hiện việc cập nhật trọng số với mỗi mẫu dữ liệu $x^{(i)}$ có nhãn tương ứng $y^{(i)}$ như sau:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J\left(\theta_t; x^{(i)}; y^{(i)}\right)$$

Với cách cập nhật này, SGD thường nhanh hơn Batch GD và có thể sử dụng để học trực tuyến (online learning) khi tập dữ liệu huấn luyện được cập nhật liên tục. Với SGD, bộ trọng số θ được cập nhật thường xuyên hơn so với Batch GD và vì vậy hàm mất mát cũng dao động nhiều hơn. Sự dao động này khiến SGD có vẻ không ổn định nhưng lại có điểm tích cực là nó giúp di chuyển đến những điểm cực tiểu (địa phương) mới có tiềm năng hơn. Với tốc độ học giảm, khả năng hội tụ của SGD cũng tương đương với Batch GD.

Cách tiếp cận thứ ba là thuật toán **Mini-batch Gradient Descent** (Mini-batch GD). Khác với hai thuật toán trước, Mini-batch GD sử dụng k điểm dữ liệu để cập nhật bộ trọng số (1 <k< N với N là tổng số điểm dữ liệu).

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J\left(\theta_t; x^{(i:i+k)}; y^{(i:i+k)}\right)$$

Mini-batch GD giảm sự dao động của hàm mất mát so với SGD và chi phí tính gradient với k điểm dữ liệu là chấp nhận được. Mini-batch GD thường được lựa chọn khi huấn luyện mạng nơron và vì vậy trong một số trường hợp, SGD được hiểu là Mini-batch GD. Riêng bản thân Mini-batch GD không đảm bảo tìm được điểm cực tiểu của hàm mất mát mà bên cạnh đó các yếu tố như tốc độ học, thuộc tính dữ liệu và tính chất của hàm mất mát cũng ảnh hưởng đến điều này.

Các tham số quan trọng trong Gradient Descent:

Trong Gradient Descent, có một số tham số quan trọng cần được xác định để điều chỉnh quá trình tối ưu hóa. Dưới đây là các tham số quan trọng trong Gradient Descent:

- Learning rate (tỷ lệ học tập): Learning rate là tham số quyết định sự di chuyển của tham số theo hướng âm gradient. Nó xác định tốc độ cập nhật của tham số. Nếu learning rate quá lớn, quá trình tối ưu hóa có thể không hội tụ và dao động. Ngược lại, nếu learning rate quá nhỏ, quá trình hội tụ có thể rất chậm. Điều chỉnh learning rate là một nhiệm vụ quan trọng để đạt được sự hội tụ và hiệu quả tối ưu.
- Momentum (đà): Momentum là một tham số được sử dụng trong các biến thể Momentum-based Gradient Descent. Nó xác định tỷ lệ của đà được tích lũy từ gradient trước đó để giúp tăng tốc độ hội tụ và tránh rơi vào các điểm cực tiểu cục bộ. Giá trị momentum thường được đặt trong khoảng từ 0 đến 1, trong đó 0 cho đà không ảnh hưởng và 1 cho đà hoàn toàn được tích lũy.
- Batch size (kích thước batch): Batch size là số lượng mẫu dữ liệu được sử dụng để tính toán gradient trong quá trình Gradient Descent. Nếu batch size bằng với tổng số mẫu dữ liệu, ta có Batch Gradient Descent. Nếu batch size bằng 1, ta có Stochastic Gradient Descent. Kích thước batch ảnh hưởng đến độ chính xác của gradient và tốc độ hội tụ. Batch size nhỏ hơn thường giúp tăng tốc độ hội tụ nhưng có thể gây nhiễu trong quá trình tối ưu hóa.
- Number of iterations (số lần lặp): Số lần lặp xác định số lần cập nhật tham số trong quá trình tối ưu hóa. Thông thường, số lần lặp được xác định trước hoặc dừng lại khi đạt được tiêu chí dừng như đạt được giá trị mất mát nhỏ đủ. Số lần lặp cần đủ để đạt được sự hội tụ đáng tin cậy của quá trình tối ưu.

1.2 Momentum

Momentum là một phương pháp tối ưu hóa dựa trên gradient descent (đạo hàm) mà nó thêm vào một phần trăm của vector cập nhật trước đó vào vector cập nhật hiện tại để tăng tốc quá trình học. Một cách đơn giản, momentum là một cách để làm mịn các cập nhật tham số mô hình và cho phép trình tối ưu hóa tiếp tục tiến về cùng hướng như trước đó, giảm thiểu sự dao động và tăng tốc độ hội tụ.

Momentum có thể được mô tả cụ thể hơn như là trung bình di động có trọng số theo cấp số nhân của gradient trước đó. Thay vì cập nhật tham số với gradient hiện

tại, trình tối ưu hóa sử dụng trung bình di động có trọng số theo cấp số nhân của các gradient trước đó. Trung bình di động có trọng số theo cấp số nhân hoạt động như một bộ nhớ cho trình tối ưu hóa, cho phép nó nhớ hướng mà nó đã di chuyển và tiếp tục di chuyển theo hướng đó, ngay cả khi gradient hiện tại chỉ điểm vào hướng khác.

Momentum được sử dụng rộng rãi kết hợp với các kỹ thuật tối ưu hóa khác như stochastic gradient descent (SGD) và các phương pháp tối ưu hóa tốc độ học thích nghi như Adagrad, Adadelta và Adam.

Gradient descent ước tính gradient của hàm mất mát đối với các tham số của mô hình ở mỗi vòng lặp và cập nhật các tham số theo hướng ngược lại với gradient, làm cho hàm mất mát giảm đi. Tốc độ học điều chỉnh độ lớn của các cập nhật tham số, và thường được đặt thành giá trị thấp để đảm bảo quá trình tối ưu hóa tiến về phía trước một cách chậm rãi.

Tuy nhiên, có nhược điểm đáng kể của gradient descent có thể làm chậm quá trình tối ưu hóa. Một trong những nhược điểm chính của gradient descent là nó có thể bị mắc kẹt tại các điểm tối thiểu cục bộ cạn hoặc các điểm xử đạt khi gradient gần bằng không và quá trình tối ưu hóa dừng lại. Một nhược điểm khác là gradient descent có thể dao động giữa các hướng đối diện, làm chậm tốc độ hội tụ.

Momentum giải quyết những nhược điểm này của gradient descent bằng cách bao gồm một yếu tố đà trong quá trình cập nhật. Thành phần đà là một phần của vector cập nhật trước đó, hoạt động như một "quả cầu trượt xuống". Thành phần đà giúp duy trì trình tối ưu hóa di chuyển theo cùng hướng khi nó di chuyển xuống, ngay cả khi gradient thay đổi hướng hoặc trở thành không đổi. Điều này giảm thiểu sự dao động và ngăn bạn bị mắc kẹt tại các điểm tối thiểu cục bộ.

Công thức cập nhật cho Momentum có thể được viết như sau:

$$\mathbf{v} = \beta\,\mathbf{v} + (1-\beta)\mathbf{abla}\,\theta\,\mathbf{J}(\theta)$$

$$\theta = \theta - \alpha\,\mathbf{v}$$

- Biến v trong công thức đại diện cho thành phần đà (momentum term), là hệ số đà (momentum coefficient)
- J(θ) là đạo hàm của hàm mất mát đối với các tham số

- α là tốc độ học (learning rate) trong phương trình này. Thông thường, hệ số đà được đặt là 0.9.

Trình tối ưu hóa tính toán gradient của hàm mất mát ở mỗi vòng lặp và cập nhật thành phần đà dưới dạng trung bình di động có trọng số theo cấp số nhân của các gradient trước đó. Sau đó, các tham số được cập nhật bằng cách trừ đi thành phần đà nhân với tốc độ học. Momentum trong machine learning giúp tăng tốc quá trình tối ưu hóa, giảm sự dao động và tránh bị mắc kẹt tại các điểm tối thiểu cục bộ. Điều này đồng nghĩa với việc hội tụ nhanh hơn, giảm thiểu dao động, đảm bảo tiến về tối thiểu toàn cầu, và tăng khả năng chống nhiễu cho các gradient. Momentum cũng có thể kết hợp với các thuật toán tối ưu hóa khác để cải thiện hiệu suất tối ưu hóa.

1.3 Adagrad, Adadelta, AMSGrad, Adam, AdamW

1.3.1 Adagrad

Thuật toán Adagrad được Duchi J. và các cộng sự đề xuất năm 2011. Khác với SGD, tốc độ học trong Adagrad thay đổi tùy thuộc vào trọng số: tốc độ học thấp đối với các trọng số tương ứng với các đặc trưng phổ biến, tốc độ học cao đối với các trọng số tương ứng với các đặc trưng ít phổ biến.

Ký hiệu g_t là gradient của hàm mất mát tại bước t. $g_{t,i}$ là đạo hàm riêng của hàm mất mát theo θ_i tại bước t.

$$g_{t,i} = \nabla_\theta J\left(\theta_{t,i}\right)$$

Quy tắc cập nhật của Adagrad:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} \cdot g_{t,i}$$

Theo quy tắc cập nhật, Adagrad điều chỉnh tốc độ học η tại bước t tương ứng với trọng số θ_i xác định dựa trên các gradient đã tính được theo θ_i . Mẫu số là chuẩn L2 (L2 norm) của ma trận đường chéo G_t trong đó phần tử i,i là tổng bình phương của các gradient tương ứng với θ_i tính đến bước t . ε là một số dương khá nhỏ nhằm

tránh trường hợp mẫu số bằng 0. Quy tắc cập nhật trên có thể viết dưới dạng tổng quát hơn như sau:

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

Trong đó, $\odot$ là phép nhân ma trận-vectơ giữa G_t và g_t . Có thể nhận thấy rằng trong thuật toán Adagrad tốc độ học được tự động điều chỉnh. Adagrad thường khá hiệu quả đối với bài toán có dữ liệu phân mảnh. Tuy nhiên, hạn chế của Adagrad là các tổng bình phương ở mẫu số ngày càng lớn khiến tốc độ học ngày càng giảm và có thể tiệm cận đến giá trị 0 khiến cho quá trình huấn luyện gần như đóng băng. Bên cạnh đó, giá trị tốc độ học η cũng phải được xác định một cách thủ công.

1.3.2 Adadelta

Thuật toán Adadelta được Zeiler và các cộng sự đề xuất năm 2012. Adadelta là một biến thể của Adagrad để khắc phục tình trạng giảm tốc độ học ở Adagrad. Thay vì lưu lại tất cả gradient như Adagrad, Adadelta giới hạn tích lũy gradient theo cửa sổ có kích thước w xác định. Bằng cách này, Adadelta vẫn tiếp tục học sau nhiều bước cập nhật. Trong quá trình thực hiện, thay vì lưu trữ w bình phương của gradient theo cách thông thường, Adadelta thực hiện tích lũy dưới dạng mô-men bậc 2 của gradient:

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

Công thức trên thể hiện trung bình các gradient $E[g^2]_t$ ở bước t phụ thuộc vào trung bình các gradient $E[g^2]_t$ ở bước $t-1$ và gradient g_t ở bước t . Hệ số γ thường có giá trị 0.9 với ý nghĩa rằng gradient ở hiện tại sẽ phụ thuộc phần lớn vào gradient ở các bước trước đó. Với thuật toán Adadelta, tốc độ học hoàn toàn được thay thế bởi:

$$\frac{\sqrt{E\left[\Delta\theta^2\right]_{t-1} + \varepsilon}}{\sqrt{E\left[g^2\right]_t + \varepsilon}}$$

Trong đó:

$$E\left[\Delta\theta^2\right]_t = \gamma E\left[\Delta\theta^2\right]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

Tương tự như trường hợp gradient, công thức trên thể hiện độ biến thiên Δθ của θ tại bước t phụ thuộc vào độ biến thiên của θ tại bước t −1.

Adadelta được cập nhật theo quy tắc:

$$\theta_{t+1} = \theta_t - \frac{\sqrt{E\left[\Delta\theta^2\right]_{t-1} + \varepsilon}}{\sqrt{E\left[g^2\right]_t + \varepsilon}} \cdot g_t$$

Adadelta không sử dụng tốc độ học. Thay vào đó, nó sử dụng tốc độ thay đổi của chính bản thân các trọng số để điều chỉnh tốc độ học.

1.3.3 Adam

Adam (Adaptive Moment Estimation) là một thuật toán cho phép tính tốc độ học thích ứng với mỗi trọng số. Adam không chỉ lưu trữ trung bình bình phương các gradient trước đó như Adadelta mà còn lưu cả giá trị trung bình mô-men m_t . Các giá trị m_t và v_t được tính bởi công thức:

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

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

trong đó β1 và β2 là các trọng số không âm, thường được chọn là 1 β = 0.9 và 2 β = 0.999 . Nếu khởi tạo m_t và v_t là các vector 0, các giá trị này có khuynh hướng nghiêng về 0, đặc biệt là khi β1 và β2 xấp xỉ bằng 1. Do vậy, để khắc phục, các giá trị này được ước lượng bằng cách:

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

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

Sau đó cập nhật các trọng số theo công thức:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t, \ \epsilon \text{ thường bằng } 10^{-8}$$

1.3.4 AMSGrad

AMSGrad sử dụng giá trị lớn nhất của các bình phương gradient trước đó v_t để cập nhật các trọng số. Ở đây, vt cũng được định nghĩa như trong thuật toán Adam:

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

Thay vì trực tiếp sử dụng v_t (hay giá trị ước lượng $\hat{v}_t$), thuật toán sẽ sử dụng giá trị trước đó v_{t-1} nếu giá trị này lớn hơn giá trị hiện tại:

$$\hat{v}_t = max(\hat{v}_{t-1}, v_t)$$

Tương tự như trong thuật toán Adam, các giá trị được ước lượng theo công thức dưới đây để khử lệch cho các trọng số:

1.3.5 AdamW

AdamW là một biến thể của Adam. Ý tưởng của AdamW khá đơn giản: khi thực hiện thuật toán Adam với L2 regularization (chuẩn hóa L2), tác giả loại bỏ phần tiêu biến của trọng số (weight decay) $w_t\theta t$ khỏi công thức tính gradient hàm mất mát tại thời điểm t :

$$g_t = \nabla f(\theta_t) + w_t \theta_t$$

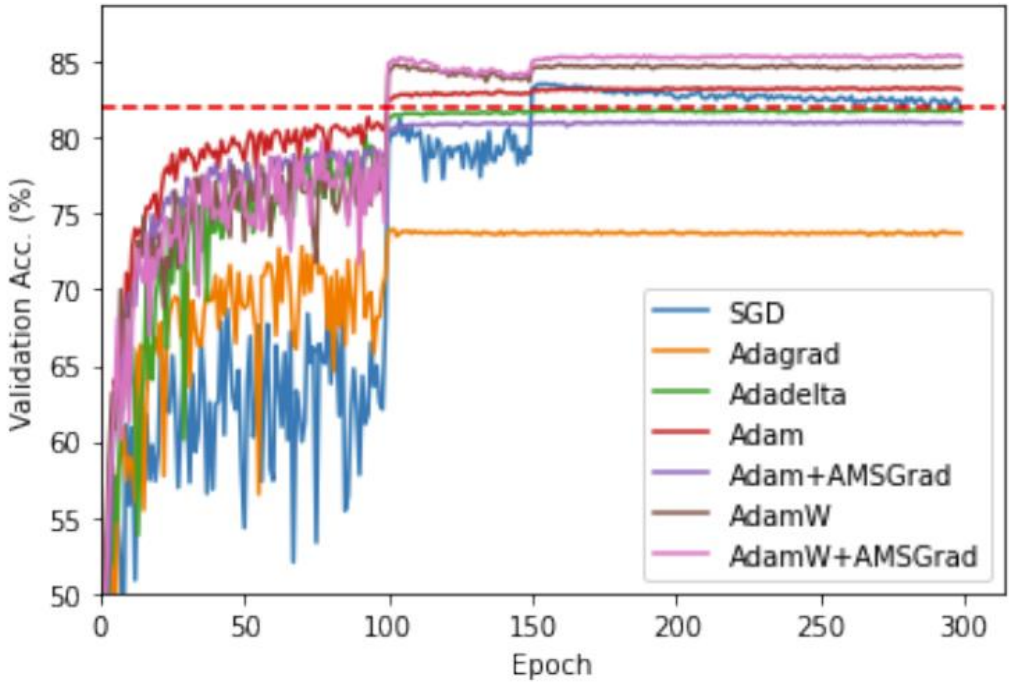
thay vào đó, đưa phần giá trị đã được phân tách này vào quá trình cập nhật trọng số:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \left(\frac{1}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t + w_{t,i} \theta_{t,i} \right), \forall t$$

Ngoài ra, trong một bài báo có đánh giá khi áp dụng những optimizer trên cho tập dữ liệu sau đây thì thu được biểu đồ và bảng kết quả thử nghiệm.

CINIC-10 là một bộ dữ liệu được sử dụng cho việc phân loại hình ảnh. Nó bao gồm tổng cộng 270,000 hình ảnh, gấp 4,5 lần so với CIFAR-10. Bộ dữ liệu này được tạo ra từ hai nguồn khác nhau: ImageNet và CIFAR-10. Cụ thể, nó được biên soạn như một liên kết giữa CIFAR-10 và ImageNet. Bộ dữ liệu này được chia thành ba tập con bằng nhau - tập huấn luyện, tập xác thực và tập kiểm tra - mỗi tập chứa 90,000 hình ảnh.

Thì thu được kết quả sau:



Hình 1: Độ chính xác của mô hình ResNet110 huấn luyện trên tập CINIC-10

Thuật toán tối ưu	Chu kỳ huấn luyện đầu tiên đạt độ chính xác kiểm thử 82%	Thời gian huấn luyện để đạt độ chính xác 82%	Thời gian huấn luyện 300 chu kỳ	Tần suất lỗi sau 300 chu kỳ (%)
SGD	151	3:28:42	**6:42:47**	18.13
Adagrad	N/A	N/A	7:10:11	26.31
Adadelta	N/A	N/A	8:10:18	18.28
Adam	**101**	2:44:58	7:33:04	16.83
Adam+AMSGrad	N/A	N/A	7:50:41	19.03
AdamW	**101**	**2:31:51**	7:10:50	15.27
AdamW+AMSGrad	**101**	2:40:57	8:00:03	**14.72**

- Thuật toán SGD có thời gian huấn luyện thấp, tốc độ hội tụ chậm nhưng ổn định, và trong thực nghiệm đều đạt được ngưỡng độ chính xác đề ra. Tuy SGD luôn đem lại một trong các tần suất lỗi nhỏ nhất, dựa vào biểu đồ có thể thấy khi quá trình huấn luyện tương đối dài, độ chính xác trên tập kiểm thử có xu hướng giảm. Để khắc phục hiện tượng này, có thể đưa thêm một số mốc chu kỳ lớn hơn vào phương án lập lịch tốc độ học. Điều này cũng chứng minh rằng hiệu quả của thuật toán SGD phụ thuộc rất lớn vào việc tinh chỉnh thủ công tốc độ học.

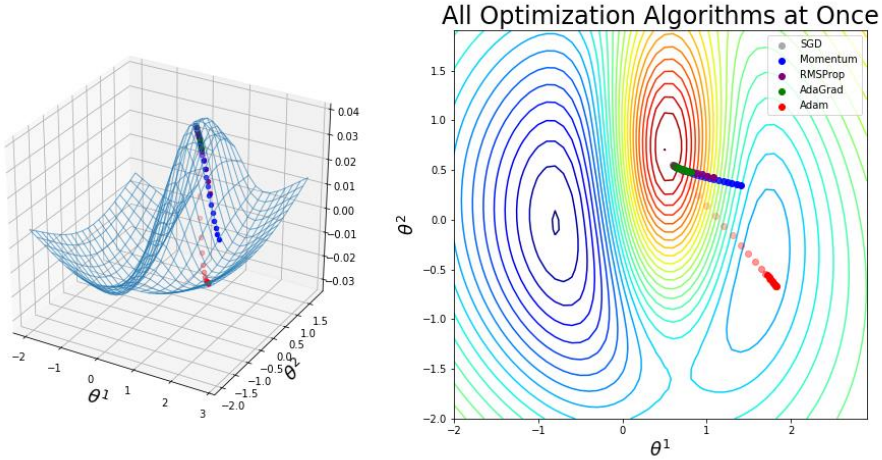
- Thuật toán Adagrad và Adadelta yêu cầu thời gian huấn luyện lớn hơn SGD, trong đó thời gian huấn luyện khi sử dụng Adadelta là lớn nhất trong các thuộc toán được so sánh. Tuy nhiên, hai thuật toán này không đem lại kết quả tương xứng: đều không đạt được ngưỡng độ chính xác đề ra trong cả hai thực nghiệm. Do đó, chúng tôi không khuyến khích sử dụng hai thuật toán này.
- Thuật toán Adam hội tụ nhanh nhưng không đạt được ngưỡng độ chính xác đề ra. Chúng tôi khuyến nghị chỉ sử dụng Adam nếu thời gian huấn luyện hạn chế và không yêu cầu quá cao về tính chính xác của mô hình.
- Biến thể của AdamW của Adam kết hợp cả ưu điểm của Adam và SGD. AdamW có thời gian huấn luyện không quá lớn so với SGD hay Adam, đem lại tốc độ hội tụ nhanh như Adam và tần suất lỗi thấp như SGD. Cũng như Adam, AdamW kế thừa khả năng tự động tinh chỉnh tốc độ học của mình trong quá trình huấn luyện. Với những ưu điểm trên, chúng tôi đề xuất sử dụng AdamW như một thuật toán lý tưởng thay thế cho SGD.
- Việc tích hợp AMSGrad vào Adam không giúp cải thiện khả năng hội tụ. Trong mọi trường hợp, AMSGrad khiến thời gian huấn luyện tăng lên đáng kể. Đối với AdamW, AMSGrad đem lại kết quả không đồng nhất. Do đó, chúng tôi cho rằng nên thận trọng khi tích hợp AMSGrad vào Adam hoặc AdamW.
- Việc lập lịch tốc độ học có ảnh hưởng tích cực không chỉ đến SGD mà cả các thuật toán với tốc độ học thích ứng như Adam: đa phần các thuật toán đều sớm đạt được ngưỡng độ chính xác đề ra ngay sau khi vượt qua các cột mốc chu kỳ trong lịch.

1.4 Nesterov Accelerated Gradient (NAG)

NAG là một biến thể của Momentum-based Gradient Descent. Nó cải thiện việc tính toán gradient bằng cách ước lượng vị trí tiếp theo của tham số sử dụng đà từ gradient trước đó. NAG thường mang lại hiệu suất tối ưu tốt hơn so với Momentum-based Gradient Descent.

Nesterov Accelerated Gradient đang cách mạng hóa máy học bằng cách giải quyết một số hạn chế của các thuật toán gradient descent truyền thống. Khả năng hội tụ nhanh hơn, xử lý gradient ồn ào và thoát khỏi điểm ngã đã biến nó thành một kỹ thuật tối ưu hóa mạnh mẽ cho huấn luyện mạng nơ-ron sâu. Khi trí tuệ nhân tạo tiếp tục phát triển, sự kết hợp của Nesterov Accelerated Gradient được dự kiến sẽ nâng cao hiệu suất của các thuật toán máy học, từ đó tạo ra những đột phá mới trong các ngành công nghiệp khác nhau.

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



Hình 2: Các optimizer phổ biến

1.5 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. Trong thực tế, tốc độ học cần được định thời bởi người lập trình. Hệ số γ xác định độ dài thông tin quá khứ được sử dụng khi điều chỉnh tỉ lệ theo từng tọa độ.

RMSprop là một phương pháp tối ưu hóa gradient dựa trên gradient của hàm mất mát được sử dụng cho việc huấn luyện các mạng nơ-ron học sâu.RMSprop được thiết kế để cập nhật trọng số của mạng nơ-ron trong quá trình huấn luyện bằng cách

thực hiện một tốc độ học tập riêng biệt cho mỗi tham số được biên thiên theo tham số t. Để giải quyết vấn đề của Adagrad, RMSprop sử dụng bằng cách chia tỷ lệ tốc độ học tập cho một trung bình bình phương của gradient.

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

Trong đó:

θ_t: tham số tại thời điểm t

η: tốc độ học tập

g_t: gradient tại thời điểm t

$E[g^2]_t$: trung bình bình phương của gradient

ϵ: một số rất nhỏ để tránh chia cho 0

$\odot$: phép nhân Hadamard

γ: hệ số chờ

CHƯƠNG 2: CONTINUAL LEARNING VÀ TEST PRODUCTION

2.1 Continual Learning

2.1.1 Giới thiệu về Continual Learning

Continual Learning (CL) là một hướng tiếp cận trong Machine Learning (ML) mà mô hình được huấn luyện trên một chuỗi các tác vụ (task) liên tục. Mỗi tác vụ có thể là một bài toán khác nhau hoặc là một phần của một bài toán lớn. Mục tiêu của CL là giúp mô hình có thể học được nhiều tác vụ khác nhau mà không quên đi những gì đã học được từ các tác vụ trước đó. Điều này có nghĩa là mô hình phải có khả năng học được các kiến thức có tính chất chung (common knowledge) giữa các tác vụ.

Continual Learning – CL tập trung vào việc phát triển các mô hình để học các nhiệm vụ mới trong khi vẫn giữ lại thông tin từ các nhiệm vụ trước đó. CL là một lĩnh vực quan trọng trong nghiên cứu vì nó đảm bảo cho tình huống thực tế, trong đó

dữ liệu và nhiệm vụ thay đổi liên tục, và một mô hình phải thích nghi với những thay đổi này mà không quên kiến thức trước đó.

Ví dụ, khi mô hình được huấn luyện để phân loại các loại động vật, nó sẽ học được các kiến thức chung như: động vật có thể di chuyển, có thể ăn, có thể sinh sản, có thể thở, có thể phát ra âm thanh, có thể có màu sắc, có thể có hình dạng, có thể có kích thước, có thể có khối lượng, có thể có tuổi. Ngoài ra, CL còn được định nghĩa là một thuật toán thích ứng học tập từ nguồn thông tin liên tục dần có theo thời gian.

Một trong những thách thức lớn trong CL là vấn đề quên đột ngột (catastrophic forgetting), khi một mô hình được huấn luyện trên nhiều nhiệm vụ cần phải nhớ thông tin học từ các nhiệm vụ trước khi tiếp xúc với dữ liệu mới. Có nhiều kỹ thuật đã được phát triển để vượt qua thách thức này, bao gồm việc áp dụng ràng buộc (regularization) và sử dụng các mạng có bộ nhớ bổ sung (memory-augmented networks).

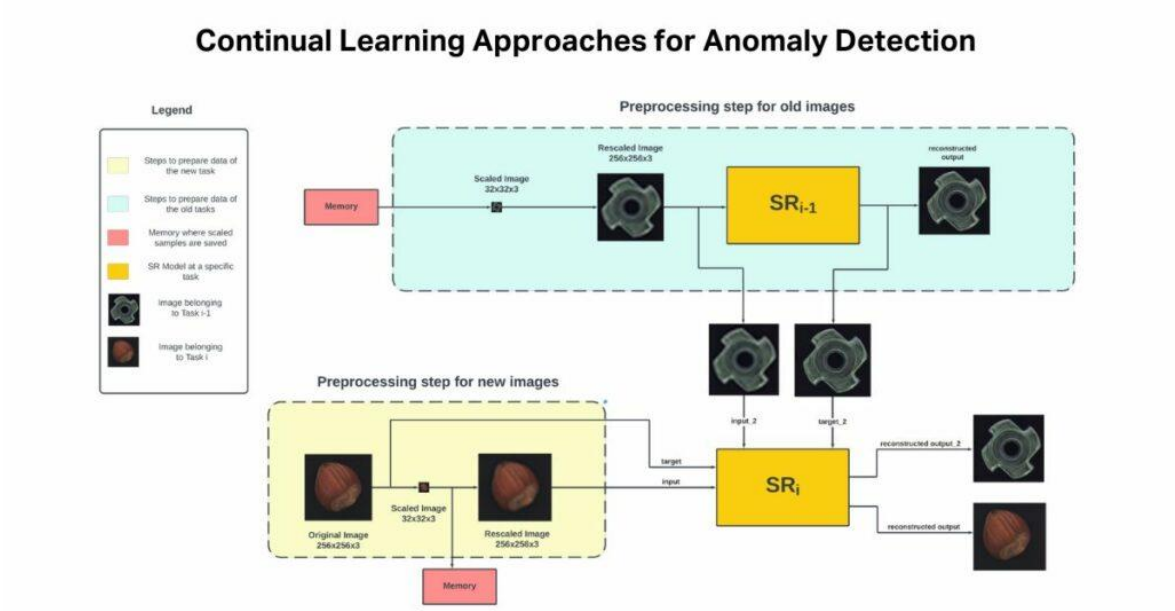
Ràng buộc liên quan đến việc thêm các ràng buộc vào quá trình học để ngăn chặn việc trang bị quá mức với dữ liệu mới. Ngược lại, các mạng có bộ nhớ bổ sung bao gồm các thành phần bộ nhớ lưu trữ thông tin từ các nhiệm vụ trước và sử dụng thông tin này để cải thiện hiệu suất trên các nhiệm vụ mới.

Sự sẵn có của dữ liệu cụ thể cho từng nhiệm vụ cũng ảnh hưởng đến tính thích nghi của mô hình. Các mô hình có truy cập đến lượng lớn dữ liệu cụ thể cho từng nhiệm vụ có khả năng thích nghi và học tốt hơn, vì họ có nhiều thông tin hơn để dự đoán. Điều này là lý do tại sao một số mô hình được huấn luyện trên lượng lớn dữ liệu để cải thiện tính thích nghi và khả năng tổng quát hóa cho các nhiệm vụ mới.

Tóm lại, CL là một lĩnh vực quan trọng trong machine learning giải quyết thách thức của việc huấn luyện các mô hình có thể liên tục học và thích nghi với các nhiệm vụ và dữ liệu mới mà không quên kiến thức trước đó. Việc sử dụng các kỹ thuật như ràng buộc và mạng có bộ nhớ bổ sung, cùng với thiết kế kiến trúc của mô hình, đóng vai trò quan trọng trong tính thích nghi của các mô hình CL.

Continual learning thường misinterpreted:

- Continual learning KHÔNG liên quan đến một lớp đặc biệt của các thuật toán ML cho phép cập nhật mô hình theo từng điểm dữ liệu mới. Ví dụ về lớp đặc biệt này của các thuật toán bao gồm cập nhật Bayes tuần tự và bộ phân loại KNN. Lớp thuật toán này tương đối nhỏ và đôi khi được gọi là "các thuật toán học trực tuyến."
- Khái niệm về Continual learning có thể áp dụng cho bất kỳ thuật toán ML có giám sát nào, không chỉ riêng một lớp cụ thể.
- Continual learning KHÔNG liên quan đến việc bắt đầu quá trình đào tạo lại mỗi khi có một mẫu dữ liệu mới xuất hiện. Trong thực tế, việc này rất nguy hiểm vì nó làm cho các mạng nơ-ron dễ bị ảnh hưởng bởi hiện tượng quên đột ngột.
- Hầu hết các công ty sử dụng học liên tục cập nhật mô hình của họ theo các lô nhỏ (ví dụ: mỗi 512 hoặc 1024 ví dụ). Số lượng ví dụ lý tưởng phụ thuộc vào nhiệm vụ cụ thể. Mặc dù học liên tục có thể trông giống như một nhiệm vụ của các nhà khoa học dữ liệu ở cái nhìn đầu tiên, nhưng thường thường, nó đòi hỏi một lượng công việc cơ sở hạ tầng đáng kể để kích hoạt nó.



2.1.2 Tại sao cần có Continual Learning

Lý do cơ bản là giúp mô hình của bạn đồng hành với sự thay đổi trong phân phối dữ liệu. Dưới đây là một số trường hợp sử dụng trong đó việc thích nghi nhanh chóng với sự thay đổi phân phối là quan trọng. Dưới đây là một số ví dụ:

- Trường hợp sử dụng trong đó sự thay đổi không mong đợi và nhanh chóng có thể xảy ra: Các trường hợp sử dụng như dịch vụ chia sẻ xe đạp phải đối mặt với trường hợp này. Ví dụ, có thể có một buổi hòa nhạc ở một khu vực ngẫu nhiên vào một thứ Hai ngẫu nhiên và "Monday pricing ML model" có thể không được trang bị tốt để xử lý nó.
- Trường hợp sử dụng trong đó không thể có được dữ liệu huấn luyện cho một sự kiện cụ thể. Một ví dụ về điều này là các mô hình thương mại điện tử trong ngày Black Friday hoặc một sự kiện giảm giá khác chưa từng thử trước đây. Rất khó để thu thập dữ liệu lịch sử để dự đoán hành vi người dùng trong Black Friday, vì vậy mô hình của bạn phải thích nghi trong suốt ngày.
- Trường hợp sử dụng nhạy cảm với vấn đề khởi đầu lạnh (cold start problem). Vấn đề này xảy ra khi mô hình của bạn phải đưa ra dự đoán cho một người dùng mới (hoặc đã đăng xuất) mà không có dữ liệu lịch sử (hoặc dữ liệu đã cũ). Nếu bạn không thích nghi với mô hình của mình ngay khi bạn nhận được một số dữ liệu từ người dùng đó, bạn sẽ không thể đề xuất các thông tin liên quan cho người dùng đó.

2.1.3 Khía cạnh của Coutinal Learning

Các khía cạnh quan trọng trong Continual Learning (CL) không chỉ giới hạn ở Catastrophic Forgetting, mà còn bao gồm:

Vượt xa quên lãng (Beyond Forgetting):

Không chỉ giữ lại thông tin từ các tác vụ trước đó để tránh quên lãng, mà còn sử dụng thông tin này để cải thiện hiệu suất trên các tác vụ sắp tới. Điều này liên quan đến khái niệm Transfer Learning, trong đó kiến thức từ một nhiệm vụ có thể được áp dụng để hỗ trợ học tập trên các nhiệm vụ khác.

Góc nhìn sinh học (Biological Perspective):

Tìm hiểu và rút ra kinh nghiệm từ cách não người xử lý thông tin và duy trì kiến thức theo thời gian. Nghiên cứu cơ chế sinh học của não bộ và các hệ thống học tập liên tục có thể cung cấp thông tin quan trọng để phát triển mô hình học máy mô phỏng lại cách não bộ hoạt động. Sự hiểu biết về các cơ chế sinh học có thể đóng vai trò quan trọng trong việc xây dựng các mô hình học liên tục hiệu quả và có thể áp dụng được trong nhiều ngữ cảnh. Những khía cạnh này mở rộng phạm vi của Continual Learning và đưa ra những thách thức và cơ hội mới để nghiên cứu và phát triển trong lĩnh vực này.

2.1.4 Bốn giai đoạn Coutinual Learning

Stage 1: Manual, stateless retraining

Stage 2: Fixed schedule automated stateless retraining

Giai đoạn này thường xảy ra khi các mô hình chính của một lĩnh vực đã được phát triển và do đó ưu tiên của bạn không còn là tạo ra các mô hình mới nữa, mà là duy trì và cải thiện các mô hình hiện có. Ở giai đoạn này, việc ở giai đoạn 1 đã trở nên quá khó chịu và không thể bỏ qua được. Tần suất đào tạo lại ở giai đoạn này thường dựa trên "ý thức cá nhân".

Điểm đổi hình thành giữa giai đoạn 1 và giai đoạn 2 thường là một đoạn mã mà một ai đó viết để chạy đào tạo lại không lưu trạng thái định kỳ. Việc viết mã này có thể rất dễ dàng hoặc rất khó tùy thuộc vào số lượng phụ thuộc cần được điều phối để đào tạo lại một mô hình.

Các bước cơ bản của đoạn mã này bao gồm:

- Trích xuất dữ liệu.
- Lấy mẫu dữ liệu xuống hoặc lên nếu cần.
- Trích xuất đặc trưng.
- Xử lý và/hoặc gắn nhãn để tạo dữ liệu đào tạo.
- Khởi đầu quá trình đào tạo.
- Đánh giá mô hình mới.
- Triển khai nó.

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

- Lịch trình
- Một cửa hàng mô hình để tự động phiên bản và lưu trữ tất cả các tạo vật cần thiết để tái tạo mô hình. Các cửa hàng mô hình trưởng thành bao gồm AWS SageMaker và Databrick's MLFlow.

Stage 3: Fixed schedule automated stateful training

Stage 4: Continual learning

Ở giai đoạn này, phần đào tạo tự động cố định của các giai đoạn trước được thay thế bằng một cơ chế kích hoạt đào tạo lại nào đó. Các kích hoạt có thể là:

- Dựa trên thời gian
- Dựa trên hiệu suất: ví dụ, hiệu suất đã giảm xuống dưới x%.

Bạn không luôn luôn có thể đo lường trực tiếp độ chính xác trong sản xuất, vì vậy bạn có thể cần sử dụng một số

2.1.5 Thách thức của Continual Learning:

- **Fresh data access challenge**

Nếu bạn muốn cập nhật mô hình của mình mỗi giờ, bạn cần có dữ liệu đào tạo được gắn nhãn chất lượng mỗi giờ. Đặc biệt, thách thức này trở nên quan trọng hơn khi tần suất cập nhật ngắn hơn.

- Tốc độ đưa dữ liệu vào kho dữ liệu

Nhiều công ty trích dữ liệu đào tạo từ kho dữ liệu của họ như Snowflake hoặc BigQuery. Tuy nhiên, dữ liệu từ các nguồn khác nhau được đưa vào kho dữ liệu bằng cách sử dụng các cơ chế khác nhau và ở tốc độ khác nhau.

Ví dụ, một phần dữ liệu trong kho có thể được trích xuất trực tiếp từ phương tiện truyền thời gian thực (sự kiện), nhưng một phần có thể đến từ các quy trình ETL hàng ngày hoặc hàng tuần sao chép dữ liệu từ nguồn khác. Một cách thông thường để giải quyết vấn đề này là trích dữ liệu trực tiếp từ phương tiện truyền thời gian thực cho việc đào tạo trước khi nó được gửi vào kho dữ liệu.

- Tốc độ gắn nhãn

Tốc độ mà bạn có thể gắn nhãn dữ liệu mới thường là điểm nghẽn. Những nhiệm vụ phù hợp nhất cho học liên tục là các nhiệm vụ có nhãn tự nhiên với các vòng lặp phản

hồi ngắn (càng ngắn vòng lặp phản hồi càng nhanh bạn có thể gắn nhãn). Nếu nhãn tự nhiên không dễ dàng có được trong khoảng thời gian cần thiết, bạn cũng có thể thử các kỹ thuật học giám sát yếu hoặc bán giám sát để có được chúng đúng thời hạn (với chi phí có thể làm ồn nhãn). Cuối cùng, bạn có thể xem xét việc gắn nhãn bằng cách sử dụng dịch vụ gắn nhãn nhanh và định kỳ.

- Yếu tố khác ảnh hưởng đến tốc độ gắn nhãn là chiến lược tính toán nhãn:

Bạn có thể tính toán nhãn theo lô. Các công việc lô này thường chạy định kỳ qua dữ liệu đã được gửi vào kho dữ liệu. Tốc độ gắn nhãn do đó phụ thuộc vào cả tốc độ gửi dữ liệu và tần suất công việc tính toán nhãn.

Tương tự như giải pháp ở trên, một cách thông thường để tăng tốc độ gắn nhãn là tính toán nhãn từ phương tiện truyền thời gian thực (sự kiện) trực tiếp. Quá trình tính toán theo luồng này có những thách thức riêng.

- **Data scaling challenge**

Phần tính toán đặc trưng thường yêu cầu Scaling. Scaling đòi hỏi truy cập vào thống kê dữ liệu toàn cầu như giá trị tối thiểu, tối đa, trung bình và phương sai. Nếu bạn đang sử dụng đào tạo có trạng thái, thống kê toàn cầu phải xem xét cả dữ liệu trước đó đã được sử dụng để đào tạo mô hình cộng với dữ liệu mới được sử dụng để làm mới nó. Theo dõi thống kê toàn cầu trong tình huống này có thể phức tạp. Một kỹ thuật thông thường để thực hiện điều này là tính toán hoặc xấp xỉ thống kê này một cách tăng dần khi bạn quan sát dữ liệu mới (thay vì nạp toàn bộ bộ dữ liệu vào thời gian đào tạo và tính toán từ đó).

- **Algorithm challenge**

Thách thức này xuất hiện khi bạn sử dụng một số loại thuật toán cụ thể và bạn muốn cập nhật chúng một cách nhanh chóng (ví dụ: mỗi giờ).

Các thuật toán liên quan đến việc cần truy cập vào toàn bộ bộ dữ liệu để được đào tạo theo thiết kế. Ví dụ, các mô hình dựa trên ma trận, giảm chiều và cây quyết định. Những loại mô hình này không thể được đào tạo một cách tăng dần bằng dữ liệu mới như các mô hình dựa trên trọng số khác có thể làm.

Ví dụ, bạn không thể thực hiện giảm chiều PCA một cách tăng dần. Bạn cần toàn bộ bộ dữ liệu. Thách thức này chỉ xảy ra khi bạn cần cập nhật chúng một cách nhanh chóng vì bạn không thể đợi cho thuật toán xử lý toàn bộ bộ dữ liệu.

Có một số biến thể của các mô hình bị ảnh hưởng đã được thiết kế để được đào tạo một cách tăng dần, nhưng việc áp dụng các thuật toán này không phổ biến rộng rãi. Một ví dụ là Cây Hoeffding và các biến thể con của nó.

- **Catastrophic Forgetting (Quên nặng nề)**

Nguy cơ quên lãng khi mô hình học từ các tác vụ mới và quên đi kiến thức đã học từ các tác vụ trước.

Một trong những thách thức chính trong việc CL là ngăn chặn đi tình trạng quên lãng. Khi mô hình được huấn luyện trên một tác vụ mới, nó có thể quên đi những gì đã học được từ các tác vụ trước đó. Điều này dẫn đến việc mô hình không thể đưa ra được các dự đoán chính xác trên các tác vụ đã được học trước đó.

- **Replay Buffer và Regularization Methods:**

Sử dụng Replay Buffer để lưu giữ một số dữ liệu từ các tác vụ trước đó và giải quyết vấn đề quên lãng. Sử dụng các phương pháp chính quy hóa như Elastic Weight Consolidation (EWC) và Synaptic Intelligence (SI) để giữ lại những gì đã học được từ các tác vụ trước đó.

Để giải quyết vấn đề này có thể sử dụng một số phương pháp như: Regularization

2.2 Test in Production

2.2.1 Giới thiệu production và test in production

Production trong ngữ cảnh của Machine Learning (ML) có thể được hiểu là việc triển khai mô hình ML vào môi trường thực tế. Điều này có thể được thực hiện bằng cách sử dụng một số phương pháp như:API, Web Application, Mobile Application, ... Tuy nhiên, việc triển khai mô hình ML vào môi trường thực tế không phải là điều dễ dàng.

- API: Một cách phổ biến để triển khai mô hình là tạo ra một API (Application Programming Interface) cho mô hình. API cho phép các ứng dụng khác truy

cập vào mô hình và nhận kết quả dự đoán. Điều này thường được sử dụng trong các hệ thống web và ứng dụng di động.

- Ứng dụng web và di động: Triển khai mô hình qua ứng dụng web hoặc di động là một cách để tích hợp mô hình ML vào các sản phẩm và dịch vụ tiếp cận người dùng cuối.
- Containerization và Microservices: Sử dụng công nghệ container như Docker và kiến trúc microservices giúp dễ dàng triển khai và quản lý các mô hình ML trong môi trường sản xuất.

Triển khai mô hình ML đòi hỏi sự hợp tác giữa các nhóm phát triển ML, quản trị hệ thống, và chuyên gia kinh doanh để đảm bảo tích hợp suôn sẻ và hiệu quả trong môi trường thực tế.

Kiểm thử trên môi trường production là 1 việc cần thiết nếu bạn muốn kiểm thử phần mềm một cách chặt chẽ nhất có thể.

Mặc dù kiểm thử sớm trong đường ống (nghĩa là kiểm thử shift-left) là cần thiết và được khuyến khích mạnh mẽ, nhưng đơn giản là nó vẫn không thực sự đủ để đảm bảo được chất lượng phần mềm là hoàn hảo. Các công ty thực hiện các phương pháp kiểm thử agile và xây dựng nên 1 cơ sở hạ tầng dùng một lần đã sẵn sàng để thực hiện kiểm thử trong môi trường production, nó đôi khi được gọi là kiểm thử shift-right.

Bằng việc kiểm thử trong môi trường production, bạn sẽ tạo nên một mức độ tự tin khác trong các bản release sau khi thực hiện các kiểm tra khác nhau trong một môi trường live production. Kiểm thử trong môi trường production cho phép công ty thấy được một ứng dụng phản ứng thế nào với khi mà có code mới được đẩy vào trong thế giới người dùng thực. Nó sẽ trở thành một thành phần quan trọng của chiến lược chất lượng ứng dụng tương lai của bạn trong tương lai.

2.2.2 Ưu và nhược điểm của việc kiểm thử trong môi trường production

a) Ưu điểm

- Các chương trình Beta nơi mà khách hàng cung cấp các phản hồi sớm về các tính năng mới và trải nghiệm người dùng.

- Ngăn chặn các thảm họa với kiểm thử phục hồi và khả năng phục hồi tốt hơn. Ứng dụng có thể phục hồi từ các sự kiện mong đợi (hỗn loạn) hoặc không mong đợi mà không mất chức năng và dữ liệu.
- Thiết kế và xây dựng 1 quy trình khắc phục thảm họa sẽ giúp giải phóng các hỗn loạn trong môi trường pre-production trước khi thực hiện trong môi trường live production.
- Bạn đang kiểm thử với dữ liệu production. (thật khó để mô phỏng theo lưu lượng và dữ liệu trên môi trường production, dẫn tới khó có thể phát hiện ra mọi tình huống có thể xảy ra để kiểm thử).
- Nó sẽ loại bỏ rủi ro của việc phát triển thường xuyên trên môi trường production khi được thực hiện hàng ngày, trong khi bạn giám sát hiệu suất ứng dụng trong thời gian thực với các công cụ như New Relic. (đại loại là nếu như có sự kiểm thử thường xuyên trên môi trường product thì việc phát hiện ra lỗi trước khi mà khách hàng báo lỗi là cao hơn)

b) Nhược điểm:

- Không có kế hoạch dự phòng cho trường hợp ứng dụng có nguy cơ mất dữ liệu
- Không có kế hoạch rollback khi release 1 bản mới.
- Để lộ ra những lỗ hổng tiềm năng
- Không thể phục hồi sau những sự hỗn loạn bất ngờ
- Thời gian kiểm thử gây ra trải nghiệm không tốt cho người dùng

2.2.3 Test Production tại sao lại quan trọng

- Độ chính xác của mô hình: Dữ liệu thực tế có thể khác biệt so với dữ liệu huấn luyện, làm giảm độ chính xác của mô hình khi triển khai.

Ví dụ: Mô hình phân loại hình ảnh huấn luyện trên dữ liệu từ máy ảnh chuyên nghiệp, nhưng khi triển khai trên ảnh từ điện thoại di động, độ chính xác giảm.

- Hiệu suất của mô hình: Mô hình có thể không đáp ứng được yêu cầu về thời gian thực hoặc tài nguyên khi triển khai.

Ví dụ: Mô hình dự đoán giá cổ phiếu trong thời gian thực, nhưng thời gian xử lý trễ làm giảm hiệu suất.

- Tránh các cuộc tấn công bất lợi: Mô hình có thể dễ bị tấn công, làm giảm đáng kể độ tin cậy khi triển khai.

Ví dụ: Mô hình xác định gương mặt trong ảnh có thể bị tấn công bằng việc thêm nhiễm đối nghịch vào ảnh.

Trong các ví dụ này, mô hình phải được kiểm thử và điều chỉnh để đảm bảo độ chính xác và hiệu suất ổn định trong môi trường thực tế, và cũng cần thử nghiệm trước để phòng tránh các cuộc tấn công tiềm ẩn.

2.2.4 Các phương pháp kiểm thử mô hình và cách đánh giá mô hình

Phương pháp kiểm thử mô hình:

- Adversarial Testing (Kiểm thử đối nghịch): Kiểm tra khả năng của mô hình chống lại dữ liệu đối nghịch hoặc tình huống biểu đồ hiệu ứng ngược. Thử nghiệm bằng cách thêm nhiễm đối nghịch vào dữ liệu đầu vào để xem liệu mô hình có giữ được độ chính xác không.
- Invariance Testing (Kiểm thử không đổi): Kiểm tra xem mô hình có thể đưa ra các dự đoán chính xác khi dữ liệu thay đổi không?
- Direction exception testing (Kiểm thử kỳ vọng định hướng): Kiểm tra xem mô hình có thể đưa ra chính xác kỳ vọng định hướng để xác định những thay đổi về phân phối đầu vào sẽ ảnh hưởng đến đầu ra như thế nào.
- Minimum functionality testing (Kiểm thử chức năng tối thiểu): Kiểm tra chức năng tối thiểu giúp quyết định xem các thành phần mô hình riêng lẻ có hoạt động như mong đợi hay không.

Đánh giá mô hình:

- Thực hiện kiểm thử sau mỗi lần cập nhật mô hình để đảm bảo rằng sự thay đổi không làm ảnh hưởng đến hiệu suất.
- Thực hiện kiểm thử kỹ thuật trước khi triển khai mô hình vào môi trường thực tế để đảm bảo tính ổn định và độ chính xác.

- Tạo bài kiểm thử để kiểm tra từng thành phần của mô hình đồng thời và độc lập
- Viết bài kiểm thử để đối mặt với các tình huống có thể xảy ra trong tương lai, giúp ngăn chặn lỗi tiềm ẩn.

2.2.5 Các chiến thuật tiềm năng để kiểm thử trên môi trường production

- Các chiến lược triển khai:
- Blue-Green Deployment

+ Blue-Green Deployment là một quy trình triển khai liên tục nhằm giảm thời gian chết và rủi ro bằng cách có hai môi trường production giống hệt nhau, được gọi là Blue và Green.

+ Blue-Green sẽ kiểm tra để có quá trình triển khai lý tưởng:

- Liền mạch: người dùng sẽ không gặp phải bất kỳ downtime nào.
- An toàn: khả năng thất bại thấp
- Hoàn toàn có thể khôi phục: có thể hoàn tác thay đổi mà không có tác dụng bất lợi.

- Kiểm thử A/B

+ AB testing là gì? A/B Testing được hiểu đơn giản là một hình thức thử nghiệm hai phiên bản A/B trong cùng một điều kiện và đánh giá xem phiên bản nào đạt hiệu quả hơn.

+ A/B testing cho thấy những thay đổi tiềm năng, có khả năng đưa ra quyết định dựa trên dữ liệu thu được và đảm bảo tác động tích cực của nó. Đây là một phần không thể thiếu trong quy trình marketing cũng như các hình thức hoạt động kinh doanh khác.

- Chiến lược rollback tự động
- Phương pháp kiểm thử trên môi trường production:
- Kiểm thử người dùng New Relic Synthetic
- Kiểm thử chấp nhận Lightweight
- Kiểm thử tích hợp cơ sở hạ tầng
- Kiểm thử trực quan với Applitools

- Kiểm thử phục hồi thảm họa
- Sự giám sát

Hiệu năng ứng dụng trong thời gian thực với New Relic, Alerts Policies

Các mẹo khi kiểm thử trong môi trường Production

- Chia kiểm thử trên production thành các tầng:

Trong khi chúng ta nói về ý tưởng 'kiểm thử trên môi trường production', nó bao gồm việc kiểm thử các ứng dụng chạy riêng trên nền tảng production, chạy trực tiếp các thử nghiệm với mã được triển khai 100% và kiểm thử toàn bộ danh sách test server trong trung tâm dữ liệu production.

Do đó, việc kiểm tra trên môi trường production nên được chia thành các tầng để kiểm tra các khía cạnh khác nhau của môi trường production sản xuất theo những cách khác nhau.

- Lên kế hoạch kiểm thử ở thời điểm mà người dùng ít sử dụng:

Kiểm thử hiệu năng có thể có gây ra tác động lớn đến toàn bộ cơ sở người dùng. Nó có thể làm cho môi trường máy chủ không hoạt động, đó là điều không ai muốn. Chúng ta nên nghiên cứu các phân tích và xác định khi nào là thời điểm tốt nhất để lên lịch kiểm thử trên môi trường production.

- Thu thập dữ liệu lưu lượng truy cập gốc cho việc kiểm thử:

Chúng ta nên thu thập và sử dụng dữ liệu lưu lượng truy cập thực tế trong môi trường production (như các quy trình làm việc của người dùng, các tài nguyên và hành vi người dùng) để thúc đẩy việc tạo tải cho các testcase. Khi bạn thực hiện các kiểm thử trong môi trường production, bạn sẽ có 1 sự tự tin là hành vi mô phỏng đó là thật.

- Tập trung giám sát:

Trong khi chạy 1 thử nghiệm trên production, hãy luôn luôn để mắt tới các số liệu về hiệu năng của người dùng để biết được rằng liệu việc thử nghiệm có gây ra những ảnh hưởng không thể chấp nhận đến trải nghiệm người dùng hay không. Chuẩn bị tinh thần để ngừng việc kiểm thử nếu điều đó xảy ra.

- Tạo 1 trải nghiệm “Opt-in”:

Một cách tuyệt vời để kiểm tra ứng dụng sẽ hoạt động như thế nào với người dùng thực tế là có một số “opt-in” ở trong các bản phát hành tính năng mới. Điều này sẽ cho phép người dùng theo dõi và thu thập dữ liệu từ người dùng trong thời gian thực và thực hiện các điều chỉnh phù hợp với chiến lược kiểm thử nghiệm mà không lo ảnh hưởng tới trải nghiệm của họ.

TÀI LIỆU THAM KHẢO

[1]. "Machine Learning: A Probabilistic Perspective" - Kevin P. Murphy.

[2]. "Pattern Recognition and Machine Learning" - Christopher M. Bishop.

[3]. "Deep Learning" - Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

[4]. "Introduction to Machine Learning with Python" - Andreas C. Müller and Sarah Guido.

[5]. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" - Aurélien Géron.