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

DỰ ÁN CUỐI KỲ MÔN NHẬP MÔN HỌC MÁY

DỰ ÁN CUỐI KỲ

Người hướng dẫn: **GV Lê Anh Cường**
Người thực hiện: **Nguyễn Hoàng Long - 51800577**

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

DỰ ÁN CUỐI KỲ MÔN NHẬP MÔN HỌC MÁY

DỰ ÁN CUỐI KỲ

Người hướng dẫn: **GV Lê Anh Cường**
Người thực hiện: **Nguyễn Hoàng Long**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Em xin chân thành cảm ơn thầy Lê Anh Cường, giảng viên môn Nhập môn Học Máy, đã tận tâm giảng dạy và truyền đạt kiến thức cho chúng em trong suốt thời gian qua.

Thầy Cường là một giảng viên có chuyên môn cao, giàu kinh nghiệm và nhiệt huyết với nghề. Thầy luôn tận tình giảng giải những kiến thức khó, giúp chúng em hiểu bài một cách dễ dàng. Ngoài ra, thầy cũng luôn quan tâm, giúp đỡ chúng em trong học tập và cuộc sống.

Nhờ sự giảng dạy tận tình của thầy, em đã có được những kiến thức nền tảng vững chắc về Nhập môn Học Máy. Em tin rằng những kiến thức này sẽ giúp ích cho em trong tương lai.

Em xin kính chúc thầy luôn mạnh khỏe, hạnh phúc và thành công trong sự nghiệp giảng dạy.

Trân trọng cảm ơn!

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

Tôi xin cam đoan đây là sản phẩm đồ án riêng của tôi và được sự hướng dẫn của …; 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 ... tháng ... năm ...

Tác giả

(ký tên và ghi rõ họ tên)

Nguyễn Hoàng Long

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

TP. Hồ Chí Minh, ngày ... tháng ... năm ...

(ký và ghi họ tên)

Phần đánh giá của GV chấm bài

TP. Hồ Chí Minh, ngày ... tháng ... năm ...

(ký và ghi họ tên)

TÓM TẮT

Dự án cuối kỳ môn Nhập môn Học Máy sẽ trình bày các phương pháp optimization trong huấn luyện mô hình học máy, cũng như 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

CHƯƠNG 1 - TÌM HIỂU VÀ SO SÁNH CÁC PHƯƠNG PHÁP OPTIMIZATION TRONG MÔ HÌNH HỌC MÁY

Phương pháp Optimization trong mô hình học máy là rất nhiều, mỗi phương pháp đều có những đặc điểm và ưu điểm riêng. Mảng Optimization trong mô hình học máy là hết sức năng động và các nhà nghiên cứu liên tục phát triển các thuật toán và biến thể mới. Vì vậy chúng ta sẽ cùng tìm hiểu một vài phương pháp optimization thường được sử dụng trong học máy. Tuy nhiên, mức độ phổ biến của chúng khác nhau, tùy theo nhiệm vụ, tập dữ liệu và kiến trúc cụ thể.

Phương pháp optimization đầu tiên chúng ta cùng tìm hiểu là Adam – Adaptive Moment Estimation. Nó là một thuật toán tối ưu hóa kết quả các khái niệm động lượng và RMSprop. Thuật toán được D. P. Kingma and J. Ba giới thiệu bài báo "Adam: Phương pháp tối ưu hóa ngẫu nhiên" vào năm 2014. Adam được sử dụng rộng rãi trong việc đào tạo mạng lưới thần kinh (neural network) và đã trờ thành một lựa chọn phổ biến do tính hiệu quả và mạnh mẽ trong nhiều nhiệm vụ khác nhau.

Một thành phần quan trọng của thuật toán Adam liên quan đến việc duy trì hai đường trung bình động theo cấp số nhân của độ dốc. Giây phút đầu tiên m_t biểu thị mức trung bình của độ dốc, trong khi khoảng khắc thứ hai v_t biểu thị mức trung bình của độ dốc bình phương. Các đường trung bình động này được cập nhật ở mỗi bước thời gian bằng cách sử dụng tốc độ phân rã theo cấp số nhân β_1 và β_2.

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

Ở đây g_t là độ dốc tại bước thời gian t và β_1 và β_2 lần lượt là tốc độ phân rã theo cấp số nhân cho khoảnh khắc thứ nhất và thứ hai.

Để hiệu chỉnh độ lệch trong các đường trung bình động, Adam kết hợp các thuật ngữ hiệu chỉnh độ lệch, dẫn đến các giá trị được điều chỉnh m_t và v_t. Các thuật ngữ được hiệu chỉnh sai lệch này sau đó được sử dụng trong công thức cập nhật tham số để đảm bảo tính ổn định và chính xác trong quá trình tối ưu hóa. Cập nhật tham số trong Adam được tính toán dựa trên các đường trung bình động đã được hiệu chỉnh sai lệch và liên quan đến các tham số mô hình θ_t, tốc độ học tập η và hằng số ε nhỏ để ổn định số.

Adam mang lại một số lợi thế, bao gồm khả năng thích ứng với các tốc độ học khác nhau, hiệu quả trong việc xử lý độ dốc thưa thớt cũng như sự kết hợp giữa các lợi ích về động lượng và RMSprop. Nó đã chứng tỏ hiệu suất mạnh mẽ trong nhiều nhiệm vụ khác nhau và thường được sử dụng trong việc đào tạo mạng lưới thần kinh sâu. Siêu tham số trong Adam bao gồm tốc độ học tập (η), tốc độ phân rã theo cấp số nhân (β_1 và β_2) và epsilon (ε). Điều chỉnh tốc độ học tập và độ nhạy đối với siêu tham số là những điều cần cân nhắc khi sử dụng Adam và việc thử nghiệm thường là cần thiết để tìm ra các giá trị tối ưu. Trong thực tế, Adam rất phù hợp để đào tạo mạng lưới thần kinh sâu, nhưng hiệu suất của nó có thể khác nhau tùy thuộc vào nhiệm vụ và tập dữ liệu cụ thể. Các kỹ thuật điều chỉnh, chẳng hạn như giảm trọng lượng, có thể được sử dụng cùng với Adam để ngăn chặn việc trang bị quá mức.

Phương pháp optimization thứ hai chúng ta tìm hiểu sẽ là Stochastic Gradient Descent (SGD) là một thuật toán một thuật toán tối ưu hóa cơ bản được sử dụng để đào tạo các mô hình học máy. Nó là một biến thể của thuật toán giảm độ dốc xử lý từng mẫu huấn luyện riêng lẻ, khiến nó đặc biệt phù hợp với các tập dữ liệu lớn.

Mục tiêu chính của SGD là giảm thiểu hàm mất mát nhất định bằng cách điều chỉnh các tham số mô hình (trọng số và độ lệch). Trong phương pháp giảm độ dốc tiêu chuẩn, các tham số mô hình được cập nhật bằng cách trừ đi độ dốc của hàm mất đối với các tham số nhân với tốc độ học (η).

Quy tắc cho Gradient Descent:

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

θ_t: Các thông số tại bước thời gian t

$\nabla J(\theta_t)$: Độ dốc của hàm mất đối với các tham số

η: Tỷ lệ học

Quy tắc cho Mini-Batch:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t; x_i; y_i)$$

x_i: Đặc điểm đầu vào của điểm dữ liệu thứ i trong mini-batch

y_i: Mục tiêu hoặc nhãn tương ứng

Trong SGD, thay vì sử dụng toàn bộ tập dữ liệu để tính toán độ dốc ở mỗi lần lặp, chỉ một tập hợp con nhỏ hoặc thậm chí một điểm dữ liệu duy nhất được sử dụng.

Tập hợp con này được gọi là "batch" hoặc "mini-batch". Quy tắc cập nhật cho một lô nhỏ liên quan đến tốc độ học, độ dốc của hàm mất đối với các tham số cũng như các tính năng đầu vào và mục tiêu của các điểm dữ liệu trong lô nhỏ.

Việc sử dụng các mini-batch ngẫu nhiên sẽ tạo ra một mức độ ngẫu nhiên hoặc ngẫu nhiên trong quá trình tối ưu hóa. Tính ngẫu nhiên này giúp thuật toán thoát khỏi cực tiểu cục bộ và khám phá không gian tham số hiệu quả hơn. Nó cũng dẫn đến việc cập nhật các tham số mô hình nhanh hơn so với việc sử dụng toàn bộ tập dữ liệu, giảm yêu cầu về bộ nhớ. Mặc dù bản chất ngẫu nhiên có thể dẫn đến sự hội tụ nhanh hơn nhưng nó cũng tạo ra các dao động trong đường dẫn tối ưu hóa. Những dao động này có thể thuận lợi cho việc thoát khỏi điểm yên ngựa nhưng có thể yêu cầu điều chỉnh tốc độ học một cách cẩn thận. Điều chỉnh tốc độ học trong quá trình đào tạo, sử dụng lịch trình tốc độ học hoặc phương pháp tốc độ học thích ứng là một chiến lược phổ biến.

Các thách thức trong SGD bao gồm các cập nhật nhiễu do các batch điểm dữ liệu riêng lẻ hoặc nhỏ, nhu cầu điều chỉnh tốc độ học cẩn thận và đánh giá sự hội tụ, điều này ít đơn giản hơn so với giảm độ dốc hàng loạt. Các biến thể của SGD bao gồm SGD Mini-Batch, SGD trực tuyến và Batch SGD, mỗi biến thể đều có đặc điểm và trường hợp sử dụng riêng. Các mẹo thực tế khi sử dụng SGD bao gồm xáo trộn dữ liệu trước mỗi kỷ nguyên, thử nghiệm lịch trình tốc độ học, thực hiện dừng sớm để ngăn chặn tình trạng trang bị quá mức cũng như theo dõi tổn thất trong quá trình đào tạo và xác thực để đánh giá độ hội tụ. SGD vẫn là một thuật toán tối ưu hóa linh hoạt và được sử dụng rộng rãi, đặc biệt là trong các tình huống có bộ dữ liệu lớn trong đó hiệu quả xử lý các Mini-Batch hơn là có lợi.

Phương pháp optimization được đề cập tiếp theo là Root Mean Square Propagation (RMSprop) là một thuật toán tối ưu hóa tốc độ học thích ứng thường được sử dụng trong đào tạo mạng lưới thần kinh. Được giới thiệu bởi Geoffrey Hinton, nó giải quyết một số hạn chế của phương pháp giảm độ dốc ngẫu nhiên (SGD) truyền thống bằng cách điều chỉnh tốc độ học cho các tham số khác nhau. RMSprop duy trì mức trung bình di chuyển theo cấp số nhân của gradient bình phương. Tại mỗi bước thời gian t, trung bình động v_t được cập nhật bằng cách sử dụng tốc độ phân rã β_2 và gradient bình phương g_t tại bước thời gian đó. Đường trung bình động này hoạt động như một hệ số giảm chấn, làm giảm tác động của độ dốc cực lớn và mang lại sự ổn định cho quá

trình tối ưu hóa. Quy tắc cập nhật cho các tham số mô hình θ_t tại mỗi bước thời gian liên quan đến việc chia gradient g_t cho căn bậc hai của đường trung bình động v_t. Hành vi thích ứng này cho phép RMSprop điều chỉnh tốc độ học riêng cho từng tham số, giúp tăng tốc độ hội tụ và xử lý các thang độ dốc khác nhau. Ngoài ra, một hằng số nhỏ ε được thêm vào để ổn định số học.

RMSprop có hiệu quả đối với các mục tiêu không cố định trong đó tốc độ học tối ưu có thể thay đổi theo thời gian. Nó loại bỏ nhu cầu điều chỉnh thủ công tốc độ học, tự động điều chỉnh tốc độ dựa trên độ dốc lịch sử. Trong khi tốc độ phân rã β_2 và tốc độ học n là các siêu tham số cần được đặt, các giá trị chung thường gần bằng 1 đối với β_2 và nằm trong khoảng từ 0.001 đến 0.1 đối với η.

Mặc dù có những ưu điểm nhưng RMSprop rất nhạy cảm với việc lựa chọn siêu tham số. Thuật toán có thể không hoạt động tốt khi xử lý các gradient rất nhiễu, vì các gradient bình phương có thể chi phối tốc độ học. Điều quan trọng là người thực hành phải thử nghiệm và điều chỉnh các siêu tham số cho các nhiệm vụ cụ thể. RMSprop, mặc dù đã thành công nhờ các thuật toán như Adam trong một số trường hợp, nhưng vẫn là lựa chọn tối ưu hóa có giá trị cho nhiều tác vụ. Cơ chế tốc độ học thích ứng của nó góp phần hội tụ nhanh hơn, khiến nó rất phù hợp với các tình huống có độ dốc không đồng nhất hoặc các mục tiêu không cố định.

Phương pháp optimization thứ tư là Adagrad – Adaptive Gradient Algorithm, là một thuật toán tối ưu hóa được thiết kế để điều chỉnh tốc độ học của các tham số mô hình trong quá trình đào tạo. Được giới thiệu bởi John Duchi, Elad Hazan và Yoram Singer vào năm 2011, Adagrad được biết đến với cơ chế tốc độ học thích ứng, khiến nó đặc biệt hữu ích cho các tình huống có dữ liệu thưa thớt và tầm quan trọng của tính năng khác nhau.

Một thành phần chính của Adagrad liên quan đến việc duy trì tổng bình phương gradient cho mỗi tham số. Tại mỗi bước thời gian t, tổng gradient bình phương G_t được cập nhật bằng cách cộng gradient bình phương g_t tại bước thời gian đó. Sự tích lũy gradient bình phương lịch sử này cho phép Adagrad điều chỉnh tốc độ học riêng cho từng tham số.

$$G_t = G_{t-1} + g^2{}_t$$

Quy tắc cập nhật cho tham số mô hình θ_t tại mỗi bước thời gian t sau đó được tính bằng cách chia gradient g_t cho căn bậc hai của tổng bình phương gradient G_t . Một hằng số nhỏ ε được thêm vào để ổn định số học. Hành vi tốc độ học thích ứng này giúp xử lý các tính năng có tầm quan trọng và quy mô khác nhau, giúp nó hoạt động hiệu quả đối với nhiều nhiệm vụ học máy.

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

Adagrad đặc biệt thuận lợi khi xử lý dữ liệu và tính năng thưa thớt. Nó phân bổ các bản cập nhật lớn hơn cho các tính năng không thường xuyên xuất hiện, cung cấp cơ chế để xử lý hiệu quả các tình huống như vậy. Điều quan trọng là Adagrad loại bỏ nhu cầu điều chỉnh tốc độ học theo cách thủ công, tự động điều chỉnh tốc độ dựa trên độ dốc bình phương lịch sử.

Tuy nhiên, Adagrad có những cân nhắc và thách thức riêng. Bản chất tích lũy của gradient bình phương có thể dẫn đến các giá trị tăng đơn điệu, có khả năng ảnh hưởng đến tốc độ học theo thời gian. Giảm tốc độ học tập hoặc các biến thể thay thế như RMSprop có thể được sử dụng để giải quyết vấn đề này. Adagrad cũng có thể phải đối mặt với những thách thức với các mục tiêu không cố định, trong đó tốc độ học tập tối ưu có thể thay đổi theo thời gian. Trong thực tế, Adagrad là một thuật toán tối ưu hóa mạnh mẽ đã đạt được thành công trong nhiều ứng dụng học máy khác nhau. Bất chấp những điểm mạnh của nó, những người thực hành nên lưu ý đến những hạn chế của nó và xem xét các phương pháp tối ưu hóa thay thế, đặc biệt khi xử lý một số loại nhiệm vụ hoặc mục tiêu đang phát triển.

Phương pháp optimization cuối cùng được đề cập tới là Nadam – Nesterov-accelerated Adaptive Moment Estimation, là một thuật toán tối ưu hóa được Timothy Dozat giới thiệu vào năm 2016. Nadam kết hợp các yếu tố từ cả Độ dốc tăng tốc của Nesterov (NAG) và Ước tính thời điểm thích ứng (Adam) để cải thiện tính hội tụ và khái quát hóa của các thuật toán tối ưu hóa.

Tương tự như Adam, Nadam duy trì hai đường trung bình động hàm mũ, m_t cho gradient và v_t cho gradient bình phương. Các đường trung bình động này được cập nhật ở mỗi bước thời gian với tốc độ phân rã lần lượt là β_1 và β_2. Ngoài ra, Nadam áp dụng hiệu chỉnh sai lệch cho các giá trị trung bình này để bù đắp cho sai lệch khởi tạo. Một

đặc điểm khác biệt của Nadam là sự kết hợp gradient tăng tốc của Nesterov. Bản cập nhật Nesterov được áp dụng sau khi hiệu chỉnh sai lệch, điều chỉnh cập nhật tham số bằng cách tính đến thuật ngữ động lượng trước vị trí hiện tại. Sự kết hợp giữa động lượng Nesterov và tốc độ học thích ứng này góp phần cải thiện tính hội tụ của thuật toán.

$$m_t = \beta_1 \cdot m_t + (1 - \beta_1) \cdot g_t$$

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

Quy tắc cập nhật tham số trong Nadam được tính bằng cách kết hợp cập nhật Nesterov với tốc độ học thích ứng. Tốc độ học tập η , tốc độ phân rã β_1 và β_2 và hằng số nhỏ ε là các siêu tham số cần được điều chỉnh dựa trên nhiệm vụ cụ thể. Nadam được thiết kế để cung cấp các đặc tính hội tụ tốt hơn, đặc biệt trong các tình huống có độ dốc nhiễu hoặc dữ liệu thưa thớt. Nó được hưởng lợi từ tốc độ học thích ứng, điều chỉnh tốc độ cho từng tham số riêng lẻ. Tuy nhiên, giống như các thuật toán tối ưu hóa khác, Nadam yêu cầu điều chỉnh siêu tham số cẩn thận và việc lựa chọn tốc độ học có thể ảnh hưởng đáng kể đến hiệu suất của nó.

$$m_t = m_t/(1 - \beta_1^t)$$

$$v_t = v_t/(1 - \beta_2^t)$$

Một điều cần cân nhắc với Nadam là chi phí tính toán của nó so với các thuật toán tối ưu hóa đơn giản hơn như Giảm dần độ dốc ngẫu nhiên (SGD). Mặc dù nó mang lại nhiều lợi ích nhưng những người thực hiện nên lưu ý đến các nguồn lực tính toán cần thiết, đặc biệt là trong các tình huống có hạn chế. Tóm lại, Nadam đã chứng minh tính hiệu quả trong nhiều nhiệm vụ học máy khác nhau, tận dụng cả động lực của Nesterov và tốc độ học thích ứng. Khi xem xét việc sử dụng nó, người thực hành nên tiến hành điều chỉnh siêu tham số thích hợp và lưu ý đến các cân nhắc tính toán liên quan đến thuật toán tối ưu hóa này.

Sau khi tìm hiểu năm phương pháp optimization, sau đây ta có thể tóm gọn lại như sau:

Giảm dần độ dốc ngẫu nhiên (SGD): Giảm dần độ dốc ngẫu nhiên là một thuật toán tối ưu hóa cơ bản cập nhật các tham số mô hình dựa trên độ dốc âm của hàm mất mát đối với các tham số đó. Tính đơn giản và dễ thực hiện của nó làm cho nó trở thành một lựa chọn phổ biến, đặc biệt đối với các bộ dữ liệu lớn. Tuy nhiên, SGD có thể nhạy

cảm với việc lựa chọn tốc độ học và độ hội tụ của nó có thể chậm, đặc biệt khi có gradient nhiễu.

Adam (Ước tính thời điểm thích ứng): Adam kết hợp các yếu tố động lượng và RMSprop, cung cấp tốc độ học thích ứng cho từng tham số. Nó xử lý hiệu quả các gradient thưa thớt và được biết đến với sự mạnh mẽ trong nhiều tác vụ khác nhau. Tuy nhiên, Adam yêu cầu phải điều chỉnh cẩn thận các siêu tham số và chi phí tính toán của nó có thể cao hơn so với các phương pháp tối ưu hóa đơn giản hơn.

RMSprop (Tuyên truyền bình phương gốc): RMSprop điều chỉnh tốc độ học một cách thích ứng bằng cách sử dụng đường trung bình động của gradient bình phương. Nó có hiệu quả đối với các vật kính không cố định và có thể xử lý các thang độ dốc khác nhau. Giống như Adam, RMSprop yêu cầu điều chỉnh siêu tham số và nó có thể không phù hợp với mọi tác vụ, đặc biệt là những tác vụ có độ dốc nhiễu.

Adagrad (Thuật toán gradient thích ứng): Adagrad điều chỉnh tốc độ học riêng cho từng tham số dựa trên độ dốc bình phương lịch sử. Nó hiệu quả đối với dữ liệu và tính năng thưa thớt, phân bổ các bản cập nhật lớn hơn cho các tính năng không xuất hiện thường xuyên. Tuy nhiên, độ dốc bình phương tích lũy của Adagrad có thể dẫn đến các giá trị tăng đơn điệu và có thể cần phải giảm tốc độ học tập hoặc các biến thể.

Nadam (Ước tính thời điểm thích ứng được tăng tốc bởi Nesterov): Nadam kết hợp động lượng Nesterov với tốc độ học thích ứng, nhằm cải thiện các đặc tính hội tụ. Nó kết hợp các lợi ích của tốc độ học tập thích ứng và độ dốc tăng tốc của Nesterov. Tương tự như Adam, Nadam yêu cầu điều chỉnh siêu tham số và chi phí tính toán của nó cao hơn so với các phương pháp đơn giản hơn.

Tóm lại tất cả các thuật toán tối ưu hóa, bao gồm SGD, Adam, RMSprop, Adagrad và Nadam, đều yêu cầu điều chỉnh cẩn thận các siêu tham số như tốc độ học, tốc độ phân rã và các hằng số nhỏ. Khả năng thích ứng của Adam, RMSprop, Adagrad và Nadam với các thang độ dốc khác nhau khiến chúng phù hợp với nhiều nhiệm vụ. Việc lựa chọn thuật toán tối ưu hóa phụ thuộc vào đặc điểm cụ thể của nhiệm vụ, tập dữ liệu và tài nguyên tính toán sẵn có. Việc thử nghiệm thường là cần thiết để xác định thuật toán và siêu tham số phù hợp nhất cho một vấn đề học máy nhất định.

CHƯƠNG 2 – TÌM HIỂU VỀ CONTINUAL LEARNING VÀ TEST PRODUCTION

Continual Learning, còn được gọi là Lifelong Learning, nghĩa là học tập suốt đời. Nó được xây dựng dựa trên ý tưởng học tập liên tục như con người chúng ta, nhằm tạo điều kiện cho sự phát triển tự chủ, gia tăng các kỹ năng lẫn kiến thức phức tạp.

Một hệ thống Continual Learning được định nghĩa một thuật toán thích ứng có khả năng học liên tục từ nguồn thông tin trực tiếp thế giới thực. Vì nguồn thông tin có sẵn và xuất hiện liên tục theo thời gian, trong đó bao gồm số lượng task cần học không xác định được trước. Điều quan trọng là nguồn cung cấp thông tin mới phải được diễn ra liên tục không dừng và không bị can thiệp (nghiêm trọng). Đồng thời Continual Learning cần thích ứng được với những thay đổi mà không quên kiến thức đã học trước đó.

Trong Continual Learning, chúng ta cần phải nắm rõ hai ý chính là:

- Catastrophic Forgetting: hiểu được thách thức của Catastrophic Forgetting, trong đó mô hình có xu hướng quên thông tin đã học trước đó khi dữ liệu mới được đưa vào.
- Task Boundaries: tìm hiểu về cách phát hiện và xử lý ranh giới nhiệm vụ, điều quan trọng trong kịch bản Continual Learning, nơi mà mô hình cần thích ứng với các yêu cầu/ nhiệm vụ (tasks) mới.

Catastrophic Forgetting, một hiện tượng được quan sát trong các mô hình học máy, đặc biệt là trong mạng lưới thần kinh (neural network). Mô hình có xu hướng quên hoặc ghi đè thông tin đã học trước đó khi được huấn luyện (train) trên dữ liệu mới. Sự quên lãng trong mô hình học máy được xem là thách thức nghiêm trọng, khi mô hình cần phải học liên tục từ luồng dữ liệu có sẵn và luôn xuất hiện theo thời gian thực.

Cụ thể rằng Catastrophic Forgetting đề cập đến tình huống trong đó một mô hình sau khi học từ một tập nhiệm vụ (tasks) hoặc một tập dữ liệu (dataset), mô hình sẽ thực hiện kém các nhiệm vụ đã học trước đó khi nó được đào tạo về các nhiệm vụ mới.

Vậy nguyên nhân của Catastrophic Forgetting là gì?

- Lượng thông tin cập nhật quá nhiều: khi một neural network được huấn luyện dữ liệu mới, trọng số (the weights) của một mô hình sẽ được điều chỉnh để

phù hợp thông tin mới. Tuy nhiên, điều này có thể dẫn đến những thay đổi đáng kể đối với cách biểu diễn đã học, dẫn đến quên đi các mẫu (patterns) đã học trước đó.

- Ghi đè thông tin: quá trình huấn luyện (train) mô hình về các nhiệm vụ mới có thể ghi đè lên các tham số (parameters) quan trọng đối với hiệu suất của nhiệm vụ trước, dẫn tới gây mất kiến thức.

Từ Catastrophic Forgetting có dẫn đến hạn chế nghiêm trọng khả năng áp dụng của mô hình trong các tình huống mà nó cần phải thích ứng với các nhiệm vụ mới trong khi phải giữ được kiến thức về các nhiệm vụ đã học trước đó.

Nhằm giảm thiểu được Catastrophic Forgetting, chúng ta cần phải phát hiện và xác định được Task Boundaries, đó là một thách thức khi mà lượng thông tin luôn được xuất hiện theo thời gian thực. Đặc biệt là trong thế giới thực, các tình huống nơi mà nhiệm vụ (tasks) có thể không được xác định rõ ràng. Task Boundaris thể hiện các điểm mà tại đó nhiệm vụ học tập thay đổi và mô hình thích ứng với thông tin hoặc nhiệm vụ mới. Việc phát hiện và quản lý những ranh giới (boundaries) hiệu quả là điều cần thiết để giảm thiểu tình trạng Catastrophic Forgetting, từ đó mà việc học liên tục (Continual Learning) không bị ảnh hưởng tới hiệu suất đối với các nhiệm vụ đã học trước đó.

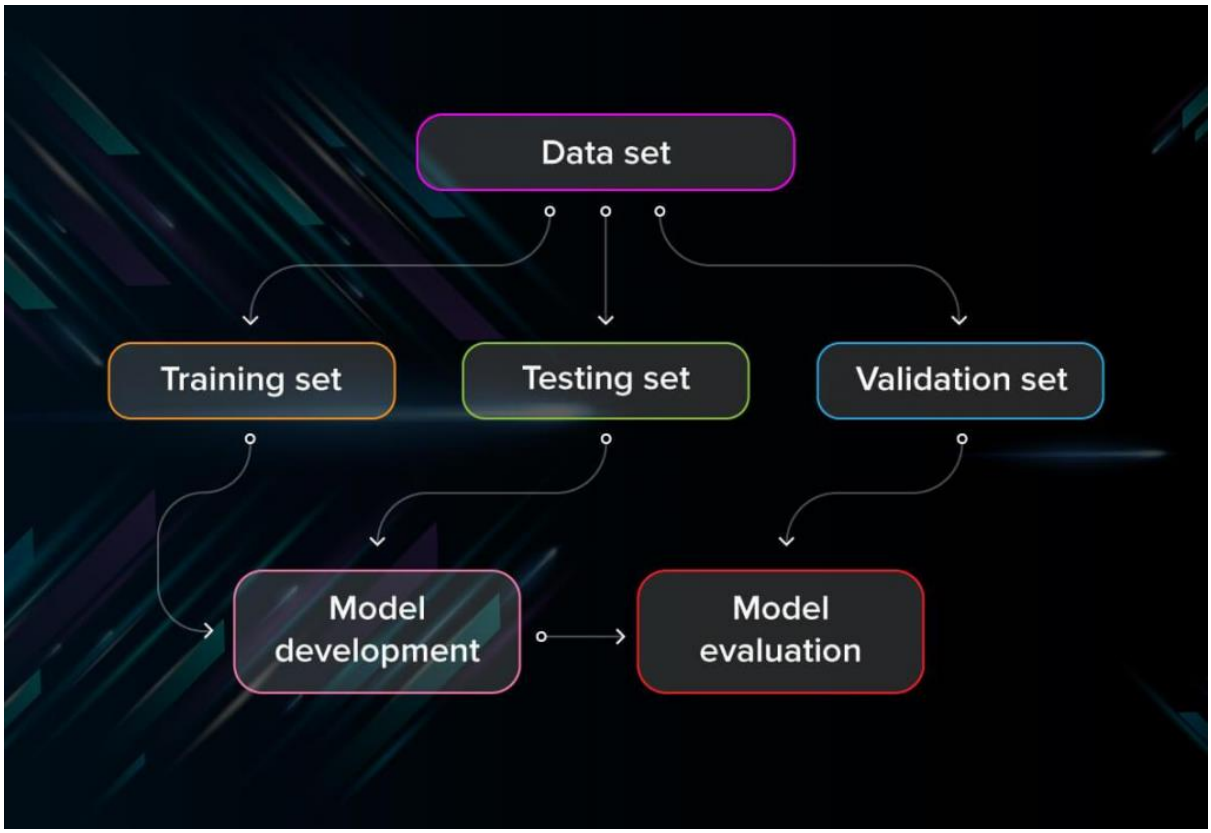
Chúng ta cần phải chú ý Dynamic Nature và Noisy Transition trong Task Boundaries. Bởi vì trong các tình huống thực tế, Task Boundaries có thể không được xác định rõ ràng hoặc không dễ nhận biết. Nhiệm vụ (tasks) có thay đổi dần dần hoặc đột ngột và mô hình cần phải điều chỉnh cho phù hợp. Quá trình chuyển đổi giữa các nhiệm vụ có thể ồn ào (noisy) hoặc mơ hồ, khiến mô hình gặp khó khăn trong việc phát chính xác khi nào nhiệm vụ mới (new task) bắt đầu.

Phương pháp để phát hiện Task Boundaries gồm có Supervised Approaches và Unsupervised Approaches. Supervised Approaches cung cấp cho mô hình thông tin được dán nhãn về thời điểm một nhiệm vụ mới bắt đầu. Unsupervised Approaches liên quan đến việc phát triển các thuật toán có thể tự động xác định Task Boundaries dựa trên những thay đổi trong phân khúc dữ liệu, hàm mất hoặc các số liệu liên quan khác.

Trước một Task Boundaries, mô hình có thể lưu trữ và phát lại các ví dụ quan trọng từ các nhiệm vụ trước đó để tránh quên. Đồng thời ta có thể dùng các kỹ thuật như Elastic Weight Consolidation (EWC) giới thiệu các thuật ngữ chính quy hóa nhằm

xử phạt các thay đổi đối với các tham số quan trọng, giảm tác động của các nhiệm vụ mới đối với kiến thức đã học trước đó.

Test Production trong học máy sẽ khác với kiểm thử phần mềm "thông thường" và đánh giá hiệu quả của một mô hình là chưa đủ.



Trong một giải pháp học máy để giải quyết bài toán nào đó, Test Production thường hay gặp một vài vấn đề như thiếu minh bạch vì một vài mô hình có thể hoạt động như "hộp đen", khiến việc hiểu hoạt động bên trong nó gặp khó khăn; Không chỉ vậy, một vài mô hình thường dựa vào thuật toán ngẫu nhiên, dẫn đến các kết quả khác nhau sau khi (re)training.

Không giống như phát triển phần mềm truyền thống, không có cách nào được thiết lập để thể hiện phạm vi thử nghiệm của mô hình học máy. Và việc thử nghiệm liên tục các mô hình học máy đòi hỏi nguồn lực (tài nguyên) và thời gian rất nhiều.

Vì vậy, Test Production trong học máy, chúng ta có thể thử nghiệm vài cách như:

- Thử nghiệm Adversarial Attack: thử nghiệm giúp phát hiện các cuộc tấn công đối nghịch bằng cách đánh giá các mô hình với các ví dụ đối nghịch trước khi triển khai, nâng cao tính mạnh mẽ của chúng.

- Thử nghiệm Data Integrity and Bias: dữ liệu được thu thập có thể phản ánh thành kiến của con người, điều này có thể được các mô hình học được trong quá trình đào tạo. Kiểm tra là rất quan trọng để xác định và giải quyết sự thiên vị, đặc biệt khi việc đánh giá chủ yếu tập trung vào các số liệu hiệu suất.
- Thử nghiệm Spot Failure Modes: kiểm tra giúp xác định các chế độ lỗi có thể xảy ra trong quá trình triển khai hệ thống máy học. Những lỗi này có thể bao gồm sai lệch hiệu suất, các vấn đề về độ bền hoặc lỗi đầu vào của mô hình mà các chỉ số đánh giá có thể bỏ qua.