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

BÁO CÁO CUỐI KÌ CÁ NHÂN
NHẬP MÔN HỌC MÁY

Người hướng dẫn: PGS.TS.LÊ VĂN CƯỜNG

Người thực hiện: **TRẦN THỊ ANH THƯ – MSSV: 52100489**

Lớp**: 21050401 - Khóa 25**

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 12 NĂM 2023

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

BÁO CÁO CUỐI KÌ CÁ NHÂN
NHẬP MÔN HỌC MÁY

Người hướng dẫn: PGS.TS.LÊ VĂN CƯỜNG

Người thực hiện: **TRẦN THỊ ANH THƯ – MSSV: 52100489**

Lớp**: 21050401 - Khóa 25**

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 12 NĂM 2023

LỜI CẢM ƠN

Môn học Nhập môn Học máy thực sự rất bổ ích, đã bổ trợ thêm cho chúng em rất nhiều kiến thức liên quan đến chuyên ngành của mình. Em xin cảm ơn sự quan tâm của nhà trường và khoa CNTT khi đã đưa bộ môn vào giảng dạy. Đặc biệt em xin gửi lời cảm ơn chân thành đến giảng viên bộ môn – thầy PGS.TS. Lê Anh Cường đã luôn ân cần giảng dạy, giải đáp tận tình những thắc mắc của chúng em, giúp chúng em có thể hiểu và làm bài một cách đơn giản nhất.

Bài báo cáo này là minh chứng rõ nhất cho những cố gắng, nỗ lực học tập của nhóm em. Tuy vậy sự tiếp thu kiến thức của chúng em chỉ mới ở mức cơ bản và còn nhiều thiếu sót, kính mong thầy có thể cho chúng em thêm nhận xét và góp ý để bài báo cáo có thể hoàn thiện hơn. Chúng em xin chân thành cảm ơn!

BÁO CÁO ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Chúng tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của thầy 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 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 19 tháng 10 năm 2023

Tác giả

Trần Thị Anh Thư

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

Trình bày về các vấn đề sau:

1. Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy;
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.

MỤC LỤC

DANH MỤC CÁC CHỮ VIẾT TẮT

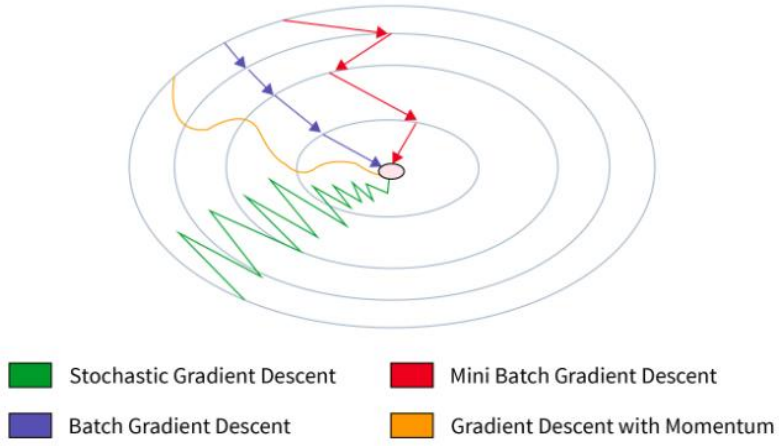
DANH SÁCH CÁC BẢNG BIỂU, HÌNH VẼ

CHƯƠNG 1
CÁC PHƯƠNG PHÁP OPTIMIZER TRONG HUẤN LUYỆN MÔ HÌNH HỌC MÁY

1. Tổng quan

Model optimization (Tối ưu hóa mô hình) đề cập đến quá trình cải thiện hiệu quả và hiệu suất của mô hình học máy mà chúng ta có thể đạt được thông qua các kỹ thuật như hyperparameter tuning (điều chỉnh siêu tham số), lựa chọn cẩn thận kiến trúc mô hình, model compression (nén mô hình), data preprocessing (tiền xử lý dữ liệu) và các chiến lược tối ưu hóa hiệu suất như sử dụng GPU, đồng thời, bộ nhớ đệm và phân batches.

Tối ưu hóa mô hình nhằm mục đích đạt được hiệu suất tốt nhất có thể từ một mô hình trong khi giảm thiểu các nguồn lực tính toán và thời gian cần thiết để đào tạo và đánh giá mô hình.



Hướng dẫn này sẽ bao gồm các trình tối ưu hóa học sâu khác nhau, chẳng hạn như Gradient Descent, Stochastic Gradient Descent, Stochastic Gradient descent with momentum, Mini-Batch Gradient Descent, Adagrad, RMSProp, AdaDelta và Adam.

Các trình tối ưu hóa phổ biến bao gồm Stochastic Gradient Descent (SGD), Adam và RMSprop. Mỗi trình tối ưu hóa có các quy tắc cập nhật, tốc độ học tập và động lượng cụ thể để tìm các thông số mô hình tối ưu nhằm cải thiện hiệu suất.

Trình tối ưu hóa là một chức năng hoặc thuật toán điều chỉnh các thuộc tính của mạng thần kinh, chẳng hạn như trọng số và Learning rate. Do đó, nó giúp giảm tổn thất tổng thể và cải thiện độ chính xác. Vấn đề lựa chọn trọng số phù hợp cho mô hình là một nhiệm vụ khó khăn, vì một mô hình học sâu thường bao gồm hàng triệu tham số.

1.1 Gradient Descent Deep Learning Optimizer:

Gradient Descent có thể được coi là “the popular kid” trong lớp tối ưu hóa. Thuật toán tối ưu hóa này sử dụng phép tính để sửa đổi các giá trị một cách nhất quán và để đạt được mức tối thiểu cục bộ. Trước khi tiếp tục, bạn có thể có câu hỏi về gradient là gì.

Nói một cách đơn giản, hãy xem xét bạn đang giữ một quả bóng nằm trên đỉnh bát. Khi bạn thả bóng, nó đi dọc theo hướng dốc nhất và cuối cùng chìm xuống đáy bát. Một Gradient cung cấp quả bóng theo hướng dốc nhất để đạt đến mức tối thiểu cục bộ là đáy bát.

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

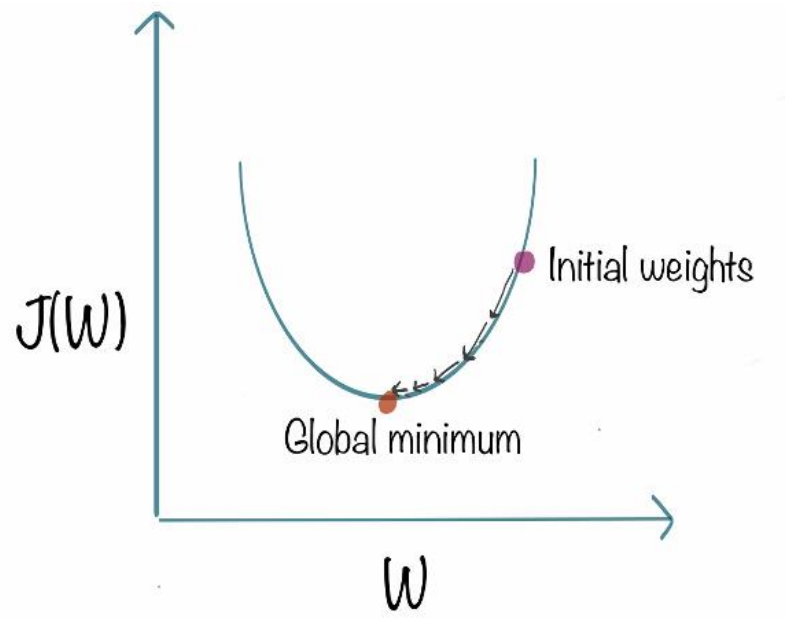
Phương trình trên có nghĩa là cách tính gradient. Ở đây alpha là kích thước bước đại diện cho khoảng cách di chuyển so với mỗi gradient với mỗi lần lặp.

Gradient descent hoạt động như sau:

- Nó bắt đầu với một số hệ số, xem chi phí của chúng và tìm kiếm giá trị chi phí thấp hơn so với hiện tại.

- Nó di chuyển về phía weights thấp hơn và cập nhật giá trị của các hệ số.

- Quá trình lặp lại cho đến khi đạt được mức tối thiểu cục bộ. Mức tối thiểu cục bộ là một điểm vượt quá mức mà nó không thể tiến hành.



Gradient descent hoạt động tốt nhất cho hầu hết các mục đích. Tuy nhiên, nó cũng có một số nhược điểm. Sẽ rất tốn kém để tính toán độ dốc nếu kích thước của dữ liệu rất lớn. Gradient descent hoạt động tốt cho các hàm lồi, nhưng nó không biết phải đi bao xa dọc theo gradient cho các hàm không lồi.

1.2 Stochastic Gradient Descent Deep Learning Optimizer

Ở cuối phần trước, bạn đã học được lý do tại sao sử dụng gradient descent trên dữ liệu lớn có thể không phải là lựa chọn tốt nhất. Để giải quyết vấn đề, chúng ta có độ dốc ngẫu nhiên. Thuật ngữ Stochastic có nghĩa là sự ngẫu nhiên mà thuật toán dựa trên. Trong stochastic gradient descent, thay vì lấy toàn bộ tập dữ liệu cho mỗi lần lặp, chúng tôi chọn ngẫu nhiên các batches dữ liệu. Lấy ngẫu nhiên dữ liệu cũng là kỹ thuật được sử dụng tránh overfitting khi train các mô hình. Điều đó có nghĩa là chúng tôi chỉ lấy một vài sample từ tập dữ liệu.

$$w := w - \eta \nabla Q_i(w).$$

Quy trình đầu tiên là chọn các tham số ban đầu w và tỷ lệ học n. Sau đó, xáo trộn ngẫu nhiên dữ liệu ở mỗi lần lặp để đạt mức tối thiểu gần đúng.

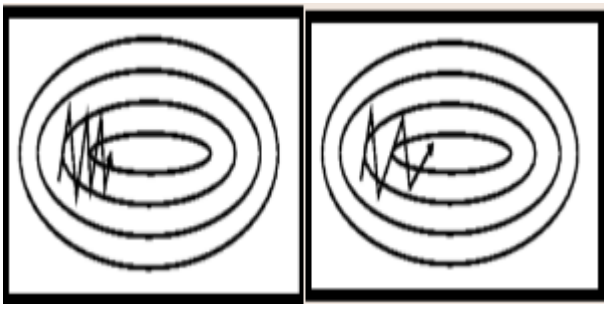
Vì chúng tôi không sử dụng toàn bộ tập dữ liệu mà là các batches của nó cho mỗi lần lặp, đường dẫn được thực hiện bởi thuật toán đầy nhiễu so với thuật toán giảm độ dốc. Do đó, SGD sử dụng số lần lặp cao hơn để đạt đến mức tối thiểu cục bộ. Do sự gia tăng số lần lặp, thời gian tính toán tổng thể tăng lên.

Nhưng ngay cả sau khi tăng số lần lặp, chi phí tính toán vẫn thấp hơn so với trình tối ưu hóa giảm độ dốc (gradient descent). Vì vậy, kết luận là nếu dữ liệu là rất lớn và thời gian tính toán là một yếu tố thiết yếu, stochastic gradient descent (giảm độ dốc ngẫu nhiên) nên được ưu tiên hơn thuật toán gradient descent hàng loạt.

1.3 Stochastic Gradient Descent với Momentum Deep Learning Optimizer

Như đã thảo luận trong phần trước, bạn đã học được rằng là nếu dữ liệu là rất lớn và thời gian tính toán là một yếu tố thiết yếu, stochastic gradient descent (giảm độ dốc ngẫu nhiên) nên được ưu tiên hơn thuật toán gradient descent hàng loạt. Vì lý do này, nó đòi hỏi một số lần lặp lại đáng kể hơn để đạt được mức tối thiểu tối ưu, và do đó thời gian tính toán rất chậm. Để khắc phục vấn đề này, chúng tôi sử dụng Stochastic Gradient Descent với thuật toán động lượng (momentum algorithm).

Những gì động lượng (momentum) làm là giúp hội tụ nhanh hơn loss function. Stochastic gradient descent dao động giữa một trong hai hướng của gradient và cập nhật weights (trọng số) cho phù hợp. Tuy nhiên, việc thêm một phần của bản cập nhật trước vào bản cập nhật hiện tại sẽ giúp quá trình nhanh hơn một chút. Một điều cần nhớ khi sử dụng thuật toán này là tốc độ học tập nên được giảm với thời hạn động lượng cao.



Trong hình trên, phần bên trái hiển thị đồ thị hội tụ của thuật toán giảm độ dốc ngẫu nhiên. Đồng thời, phía bên phải hiển thị SGD với momentum. Từ hình ảnh, bạn có thể so sánh đường dẫn được chọn bởi cả hai thuật toán và nhận ra rằng sử dụng động lượng giúp đạt được sự hội tụ (convergence) trong

thời gian ngắn hơn. Bạn có thể nghĩ đến việc sử dụng một động lực lớn và tốc độ học tập để làm cho quá trình thậm chí còn nhanh hơn. Nhưng hãy nhớ rằng trong khi tăng động lượng, khả năng vượt qua mức tối thiểu tối ưu cũng tăng lên. Điều này có thể dẫn đến độ chính xác kém và thậm chí nhiều dao động hơn.

1.4 Mini Batch Gradient Descent Deep Learning Optimizer

Trong biến thể gradient descent này, thay vì lấy tất cả trainning data, chỉ một tập hợp con của tập dữ liệu được sử dụng để tính toán loss function. Vì chúng tôi đang sử dụng một loạt dữ liệu thay vì lấy toàn bộ tập dữ liệu, nên cần ít lần lặp lại hơn. Đó là lý do tại sao thuật toán giảm độ dốc hàng loạt nhỏ (mini-batch gradient descent algorithm) nhanh hơn cả thuật toán giảm độ dốc ngẫu nhiên (stochastic gradient descent) và giảm độ dốc hàng loạt (batch gradient descent algorithms). Thuật toán này hiệu quả và mạnh mẽ hơn so với các biến thể trước đó của gradient descent. Vì thuật toán sử dụng phân lô, tất cả dữ liệu đào tạo không cần phải được tải trong bộ nhớ, do đó làm cho quá trình thực hiện hiệu quả hơn. Hơn nữa, cost function trong mini-batch gradient descent nhiễu hơn thuật toán gradient descent hàng loạt nhưng mượt mà hơn so với thuật toán gradient descent ngẫu nhiên. Bởi vì điều này, giảm độ dốc hàng loạt nhỏ là lý tưởng và cung cấp sự cân bằng tốt giữa tốc độ và độ chính xác.

Bất chấp tất cả những điều đó, thuật toán giảm độ dốc hàng loạt nhỏ cũng có một số nhược điểm. Nó cần một siêu tham số là "mini-batch-size", cần được điều chỉnh để đạt được độ chính xác cần thiết. Mặc dù, kích thước lô 32 được coi là phù hợp với hầu hết mọi trường hợp. Ngoài ra, trong một số trường hợp, nó dẫn đến độ chính xác cuối cùng kém. Do đó, cũng cần phải tăng lên để tìm kiếm các lựa chọn thay thế khác.

1.5 Adagrad (Adaptive Gradient Descent) Deep Learning Optimizer

Thuật toán giảm độ dốc thích ứng hơi khác so với các thuật toán gradient descendcent khác. Điều này là do nó sử dụng tỷ lệ học tập khác nhau cho mỗi lần lặp. Sự thay đổi tỷ lệ học tập phụ thuộc vào sự khác biệt về các thông số trong quá trình đào tạo. Các thông số càng được thay đổi, tốc độ học tập càng thay đổi nhỏ. Sửa đổi này rất có lợi vì các bộ dữ liệu trong thế giới thực chứa các tính năng thưa thớt cũng như dày đặc. Vì vậy, thật không công bằng khi có cùng giá trị về tỷ lệ học tập cho tất cả các tính năng. Thuật toán Adagrad sử dụng công thức dưới đây để cập nhật trọng số. Ở đây alpha (t) biểu thị tốc độ học tập khác nhau ở mỗi lần lặp, n là một hằng số và E là một dương nhỏ để tránh chia cho 0.

$$\mathrm{w_t} = \mathrm{w_{t-1}} - \eta'_t \frac{\partial \mathrm{L}}{\partial \mathrm{w(t-1)}} \qquad \eta'_t = \frac{\eta}{\mathrm{sqrt}(\alpha_t + \epsilon)}$$

Lợi ích của việc sử dụng Adagrad là nó loại bỏ nhu cầu sửa đổi tỷ lệ học tập theo cách thủ công. Nó đáng tin cậy hơn các thuật toán giảm độ dốc và các biến thể của chúng, và nó đạt đến sự hội tụ ở tốc độ cao hơn.

Một nhược điểm của trình tối ưu hóa AdaGrad là nó làm giảm tỷ lệ học tập một cách mạnh mẽ và đơn điệu. Có thể có một thời điểm khi tỷ lệ học tập trở nên cực kỳ nhỏ. Điều này là do các gradient bình phương trong mẫu số tiếp tục tích lũy, và do đó phần mẫu số tiếp tục tăng. Do tỷ lệ học tập nhỏ, mô hình cuối cùng không thể có thêm kiến thức và do đó độ chính xác của mô hình bị tổn hại.

1.6 RMS Prop (Root Mean Square) Deep Learning Optimizer

RMS prop là một trong những trình tối ưu hóa phổ biến trong số những người đam mê học sâu. Điều này có thể là do nó chưa được xuất bản nhưng vẫn rất nổi tiếng trong cộng đồng. RMS prop lý tưởng là một phần mở rộng của RPPROP công việc. Nó giải quyết vấn đề của các gradient khác nhau. Vấn đề với độ dốc là một số trong số chúng nhỏ trong khi những người khác có thể rất lớn. Vì vậy, xác định một tỷ lệ học tập duy nhất có thể không phải là ý tưởng tốt nhất. RPPROP sử dụng dấu hiệu gradient, điều chỉnh kích thước bước riêng

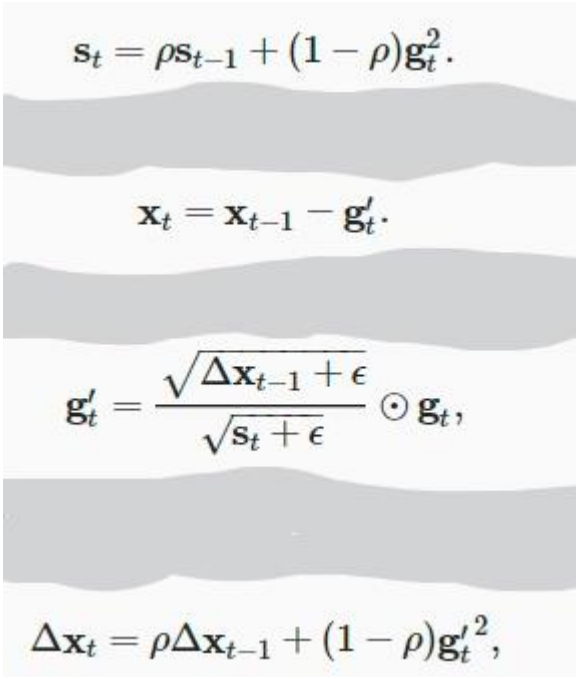
cho từng trọng lượng. Trong thuật toán này, hai gradient đầu tiên được so sánh cho các dấu hiệu. Nếu chúng có cùng một dấu hiệu, chúng ta đang đi đúng hướng, tăng kích thước bước lên một phần nhỏ. Nếu chúng có dấu hiệu ngược lại, chúng ta phải giảm kích thước bước. Sau đó, chúng tôi giới hạn kích thước bước và bây giờ có thể cập nhật trọng lượng.

Vấn đề với RPPROP là nó không hoạt động tốt với các bộ dữ liệu lớn và khi chúng tôi muốn thực hiện cập nhật hàng loạt nhỏ. Vì vậy, đạt được sự mạnh mẽ của RPPROP và hiệu quả của các lô nhỏ đồng thời là động lực chính đằng sau sự gia tăng của cánh quạt RMS. RMS prop là một tiến bộ trong trình tối ưu hóa AdaGrad vì nó làm giảm tốc độ học tập giảm đơn điệu.

1.7 AdaDelta Deep Learning Optimizer

AdaDelta có thể được xem là một phiên bản mạnh mẽ hơn của trình tối ưu hóa AdaGrad. Nó dựa trên học tập thích ứng và được thiết kế để đối phó với những nhược điểm đáng kể của trình tối ưu hóa đạo cụ AdaGrad và RMS. Vấn đề chính với hai trình tối ưu hóa trên là tốc độ học tập ban đầu phải được xác định thủ công. Một vấn đề khác là tốc độ học tập suy giảm trở nên vô cùng nhỏ tại một số điểm. Do đó, một số lần lặp lại nhất định sau đó, mô hình không còn có thể học kiến thức mới.

.



$$\mathbf{s}_t = \rho \mathbf{s}_{t-1} + (1-\rho)\mathbf{g}_t^2.$$

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \mathbf{g}'_t.$$

$$\mathbf{g}'_t = \frac{\sqrt{\Delta \mathbf{x}_{t-1} + \epsilon}}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \mathbf{g}_t,$$

$$\Delta \mathbf{x}_t = \rho \Delta \mathbf{x}_{t-1} + (1-\rho)\mathbf{g}'^2_t,$$

Để giải quyết những vấn đề này, AdaDelta sử dụng hai biến trạng thái để lưu trữ trung bình rò rỉ của gradient khoảnh khắc thứ hai và trung bình rò rỉ của thời điểm thay đổi tham số thứ hai trong mô hình. Ở đây St và delta Xt biểu thị các biến trạng thái, g't biểu thị gradient được chia tỷ lệ lại, delta Xt-1 biểu thị các hình vuông gradient được chia tỷ lệ lại và epsilon biểu thị một số nguyên dương nhỏ để xử lý phép chia cho 0

1.8 Adam Optimizer trong Deep Learning

Adam optimizer, viết tắt của Adaptive Moment Estimation optimizer, là một thuật toán tối ưu hóa thường được sử dụng trong deep learning. Nó là một phần mở rộng của thuật toán Stochastic gradient Descent (SGD) và được thiết kế để cập nhật trọng số của mạng lưới thần kinh trong quá trình đào tạo.

Cái tên "Adam" có nguồn gốc từ "ước tính thời điểm thích ứng", làm nổi bật khả năng điều chỉnh thích ứng tốc độ học tập cho từng trọng số mạng riêng lẻ. Không giống như SGD, duy trì một tỷ lệ học tập duy nhất trong suốt quá trình đào tạo, trình tối ưu hóa Adam tự động tính toán tỷ lệ học tập cá nhân dựa trên độ dốc trong quá khứ và khoảnh khắc thứ hai của chúng.

Những người tạo ra trình tối ưu hóa Adam đã kết hợp các tính năng có lợi của các thuật toán tối ưu hóa khác như AdaGrad và RMSProp. Tương tự

như RMSProp, trình tối ưu hóa Adam xem xét khoảnh khắc thứ hai của gradient, nhưng không giống như RMSProp, nó tính toán phương sai không căn giữa của gradient (mà không trừ giá trị trung bình).

Bằng cách kết hợp cả khoảnh khắc đầu tiên (trung bình) và khoảnh khắc thứ hai (phương sai không tập trung) của các gradient, trình tối ưu hóa Adam đạt được tốc độ học tập thích ứng có thể điều hướng hiệu quả bối cảnh tối ưu hóa trong quá trình đào tạo. Khả năng thích ứng này giúp hội tụ nhanh hơn và cải thiện hiệu suất của mạng lưới thần kinh.

Tóm lại, Adam optimizer là một thuật toán tối ưu hóa mở rộng SGD bằng cách tự động điều chỉnh tỷ lệ học tập dựa trên trọng số cá nhân. Nó kết hợp các tính năng của AdaGrad và RMSProp để cung cấp các bản cập nhật hiệu quả và thích ứng cho trọng số mạng trong quá trình đào tạo học sâu.

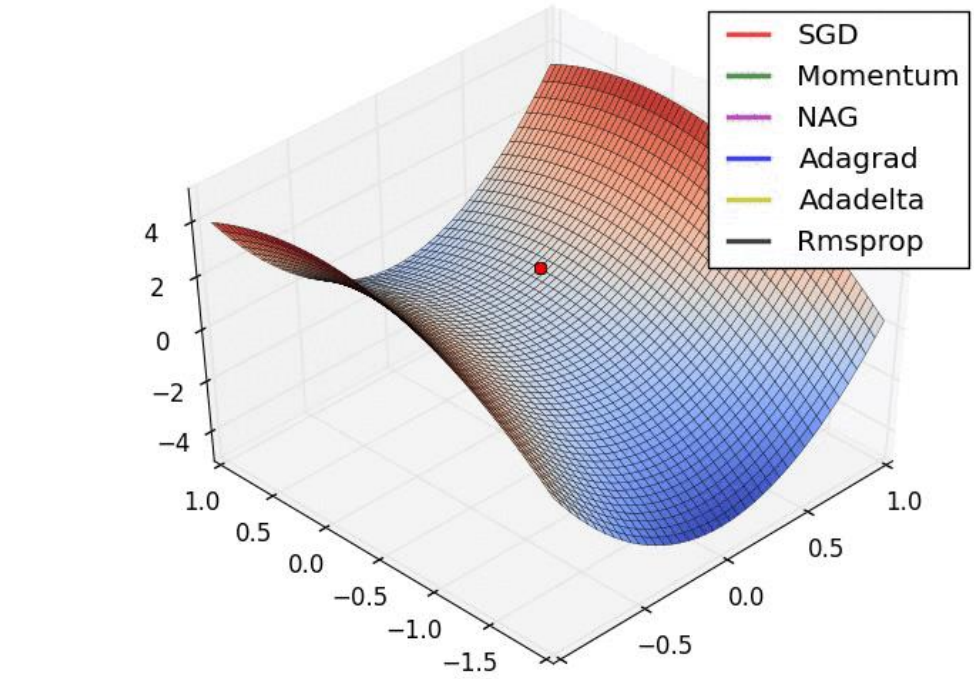
Công thức Adam Optimizer

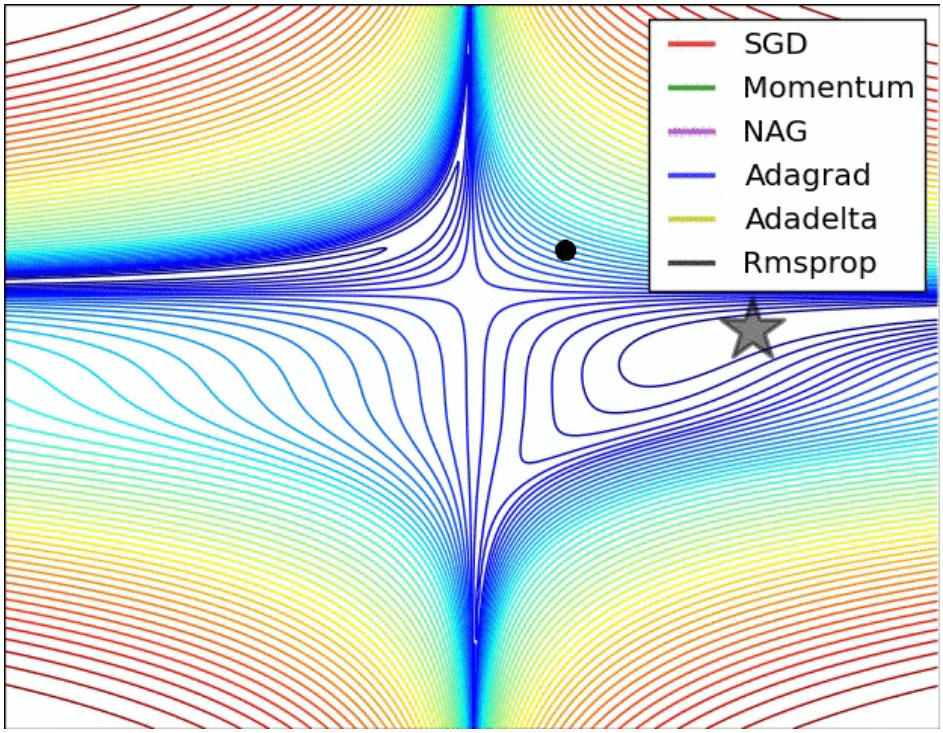
Trình tối ưu hóa adam có một số lợi ích, do đó nó được sử dụng rộng rãi. Nó được điều chỉnh như một điểm chuẩn cho các bài báo học sâu và được đề xuất như một thuật toán tối ưu hóa mặc định. Hơn nữa, thuật toán rất đơn giản để thực hiện, có thời gian chạy nhanh hơn, yêu cầu bộ nhớ thấp và yêu cầu điều chỉnh ít hơn bất kỳ thuật toán tối ưu hóa nào khác.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t}\right] v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t}\right]^2$$

Công thức trên thể hiện hoạt động của trình tối ưu hóa adam. Ở đây B1 và B2 đại diện cho tốc độ phân rã của trung bình của các gradient.

Nếu trình tối ưu hóa adam sử dụng các thuộc tính tốt của tất cả các thuật toán và là trình tối ưu hóa tốt nhất hiện có, thì tại sao bạn không nên sử dụng Adam trong mọi ứng dụng? Và nhu cầu tìm hiểu sâu về các thuật toán khác là gì? Điều này là do ngay cả Adam cũng có một số nhược điểm. Nó có xu hướng tập trung vào thời gian tính toán nhanh hơn, trong khi các thuật toán như giảm độ dốc ngẫu nhiên tập trung vào các điểm dữ liệu. Đó là lý do tại sao các thuật toán như SGD khái quát hóa dữ liệu theo cách tốt hơn với chi phí tốc độ tính toán thấp. Vì vậy, các thuật toán tối ưu hóa có thể được chọn cho phù hợp tùy thuộc vào yêu cầu và loại dữ liệu.





Các hình ảnh trực quan ở trên tạo ra một bức tranh tốt hơn trong tâm trí và giúp so sánh kết quả của các thuật toán tối ưu hóa khác nhau.

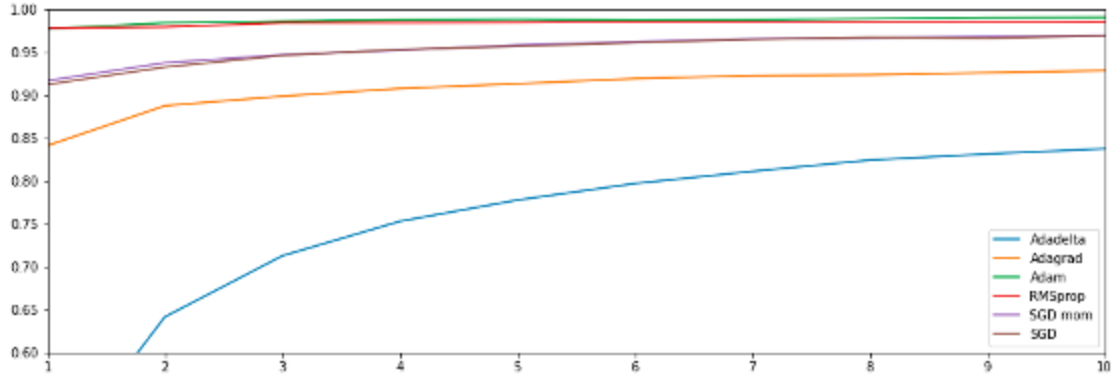
Ưu và nhược điểm của trình tối ưu hóa

Optimizer	Pros	Cons
Stochastic Gradient Descent (SGD)	- Thực hiện đơn giản và hiệu quả về mặt tính toán.. - Hiệu quả đối với các tập dữ liệu lớn với không gian đặc trưng nhiều chiều.	- SGD có thể bị kẹt ở mức cực tiểu địa phương.. - Độ nhạy cao với tốc độ học tập ban đầu..
Stochastic Gradient Descent with Gradient Clipping	- Giảm khả năng bùng nổ độ dốc. - Cải thiện sự ổn định đào tạo	- Việc cắt bớt có thể che giấu các vấn đề khác như khởi tạo kém hoặc tốc độ học kém.
Momentum	- Giảm dao động trong quá trình luyện tập. - Hội tụ nhanh hơn cho các bài toán không điều kiện.	- Tăng độ phức tạp của thuật toán.
Nesterov Momentum	- Hội tụ nhanh hơn động lượng cổ điển. - Có thể giảm hiện tượng overshooting.	- Đắt hơn classical momentum.
Adagrad	- Tỷ lệ học tập thích ứng cho mỗi tham số. - Hiệu quả đối với dữ liệu thưa thớt.	- Có thể dừng việc học quá sớm. - 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.

Optimizer	Pros	Cons
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	- Áp dụng cho các tập dữ liệu lớn và mô hình nhiều chiều.- Áp dụng cho các tập dữ liệu lớn và mô hình nhiều chiều.- Khả năng khái quát hóa tốt.	- Yêu cầu điều chỉnh cẩn thận các siêu tham số.
Adamax	- Mạnh mẽ hơn đối với không gian nhiều chiều.- Hoạt động tốt khi có độ dốc ồn.	- Đắt tiền.
SMORMS3	- Hiệu suất tốt trên các tập dữ liệu lớn với không gian nhiều chiều. - Hiệu suất ổn định khi có độ dốc ồn.	- Đắt tiền.

1.9 Train với bộ dữ liệu Mnist.csv, cho được kết quả

```
batch_size=64

num_classes=10

epochs=10

def build_model(optimizer):

model=Sequential()

model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy, optimizer= optimizer, metrics=['accur

return model
```



- Trình tối ưu hóa adam cho thấy độ chính xác tốt nhất trong một khoảng thời gian thỏa đáng.
- RMSprop cho thấy độ chính xác tương tự như của Adam nhưng với thời gian tính toán tương đối lớn hơn nhiều.
- Đáng ngạc nhiên, thuật toán SGD mất ít thời gian nhất để đào tạo và cũng tạo ra kết quả tốt. Nhưng để đạt được độ chính xác của trình tối ưu hóa Adam, SGD sẽ yêu cầu nhiều lần lặp lại hơn và do đó thời gian tính toán sẽ tăng lên.
- SGD với động lượng cho thấy độ chính xác tương tự như SGD với thời gian tính toán lớn hơn bất ngờ. Điều này có nghĩa là giá trị của động lượng được thực hiện cần phải được tối ưu hóa.
- Adadelta cho thấy kết quả kém cả về độ chính xác và thời gian tính toán.

2. Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

- Cotinual learning

  - Để điều chỉnh các mô hình cho phù hợp với sự thay đổi phân phối dữ liệu

  - Vấn đề cơ sở hạ tầng

- Test in production

  - Mô hình được đào tạo lại để thích ứng với môi trường thay đổi

    Đánh giá nó trên một bộ stationary set.

    Cũng thử nghiệm trong sản xuất

  - Monitoring & test in production

    Giám sát: Theo dõi thụ động các kết quả đầu ra

    Thử nghiệm trong sản xuất: Chủ động lựa chọn mô hình nào để tạo ra đầu ra

    Mục tiêu: để hiểu hiệu suất của mô hình và tìm ra thời điểm cập nhật nó

- Goal of continual learning

  - Để tự động hóa cập nhật một cách an toàn và hiệu quả

2.1 Continual Learning

Continual learning (học liên tục) là một lĩnh vực trong học máy và trí tuệ nhân tạo, nghiên cứu cách để mô hình máy học tiếp tục học và cải thiện hiệu suất của nó khi đối mặt với dữ liệu mới mà không cần huấn luyện lại từ đầu. Trong học liên tục, mô hình cố gắng học từ các tập dữ liệu tiếp theo mà nó nhận được sau khi đã được huấn luyện ban đầu và giữ lại các kiến thức đã học trước đó.

Mục tiêu của học liên tục là tạo ra các mô hình mà có thể liên tục học hỏi từ dữ liệu mới, hợp nhất kiến thức đã học trước đó và tránh hiện tượng quên mất kiến thức cũ khi đối mặt với dữ liệu mới. Điều này có thể hữu ích trong các tình huống thực tế, nơi dữ liệu tăng lên theo thời gian và mô hình cần cập nhật để duy trì hiệu suất cao.

```
previous_model = previous_generator = None
for context, train_dataset in enumerate(train_datasets, 1):
    if scenario=="task":
        active_classes = [list(
            range(model.classes_per_context * i, model.classes_per_context * (i+1))
        ) for i in range(context)]
    else:
        active_classes = None

    iters_left = 1
    progress = tqdm.tqdm(range(1, iters+1))

    for batch_index in range(1, iters+1):
        iters_left -= 1
        if iters_left==0:
            data_loader = iter(utils.get_data_loader(train_dataset, batch, cuda=cuda, drop_last=True))
            iters_left = len(data_loader)

        x, y = next(data_loader)
        y = y-model.classes_per_context*(context-1) if scenario=='task' else y
        x, y = x.to(device), y.to(device)
```

```
        if previous_model is not None:
            x_ = previous_generator.sample(batch, only_x=True)
            if not scenario=='task':
                with torch.no_grad():
                    scores_ = previous_model.classify(x_)
                _, y_ = torch.max(scores_, dim=1)
            else:
                scores_ = list()
                y_ = list()
                with torch.no_grad():
                    all_scores_ = previous_model.classify(x_)
                for context_id in range(context-1):
                    temp_scores_ = all_scores_[:, active_classes[context_id]]
                    scores_.append(temp_scores_)
                    _, temp_y_ = torch.max(temp_scores_, dim=1)
                    y_.append(temp_y_)
            y_ = y_ if (model.replay_targets == "hard") else None
            scores_ = scores_ if (model.replay_targets == "soft") else None
        else:
            x_ = y_ = scores_ = None
        loss_dict = model.train_a_batch(
            x, y, x_=x_, y_=y_, scores_=scores_, rnt = 1./context,
            active_classes=active_classes, context=context
        )
        # Train the generative model on this batch
        _ = generator.train_a_batch(x, x_=x_, rnt=1./context)
```

```
            # Update progress bar
            context_stm = " Context: {}/{} |".format(context, contexts)
            progress.set_description(
                '<CLASSIFIER> |{t_stm} training loss: {loss:.3} | training accuracy: {prec:.3} |'
                    .format(t_stm=context_stm, loss=loss_dict['loss_total'], prec=loss_dict['accuracy'])
            )
            progress.update(1)
            # Execute callback-function to keep track of performance throughout training
            if eval_cb is not None:
                eval_cb(model, batch_index, context=context)
        # Close progres-bar
        progress.close()
        # Update the source for the replay
        previous_generator = copy.deepcopy(generator).eval()
        previous_model = copy.deepcopy(model).eval()
```

```
<CLASSIFIER> | Context: 1/5 | training loss: 1.02e-06 | training accuracy: 1.0 |: 100%|██████████| 1000/1000 [00:28<00:00, 35.00it/s]
<CLASSIFIER> | Context: 2/5 | training loss: 0.00302 | training accuracy: 1.0 |: 100%|██████████| 1000/1000 [00:35<00:00, 27.94it/s]
<CLASSIFIER> | Context: 3/5 | training loss: 0.0424 | training accuracy: 1.0 |: 100%|██████████| 1000/1000 [00:36<00:00, 27.69it/s]
<CLASSIFIER> | Context: 4/5 | training loss: 0.0366 | training accuracy: 1.0 |: 100%|██████████| 1000/1000 [00:36<00:00, 27.38it/s]
<CLASSIFIER> | Context: 5/5 | training loss: 0.14 | training accuracy: 0.977 |: 100%|██████████| 1000/1000 [00:37<00:00, 26.70it/s]
```

Huấn luyện bộ phân loại và mô hình tổng quát về thí nghiệm học liên tục giải quyết với bài toán Mnist.csv.

2.2 Test Production

Về phần thử nghiệm các mô hình máy học trên môi trường sản xuất (production), quá trình này liên quan đến triển khai mô hình đã được huấn luyện vào một hệ thống hoạt động thực tế. Điều này bao gồm việc đưa mô hình vào môi trường sản xuất, tích hợp với hệ thống, xử lý dữ liệu đầu vào và đầu ra, và đánh giá hiệu suất thực tế của mô hình.

Trước khi triển khai mô hình vào môi trường sản xuất, các bước sau có thể được thực hiện:

- Chọn mô hình tốt nhất: Đánh giá và chọn mô hình tốt nhất dựa trên độ chính xác và hiệu suất trên tập kiểm tra hoặc xác thực.

- Chuẩn bị dữ liệu: Đảm bảo dữ liệu đầu vào của mô hình trong môi trường sản xuất tuân thủ các yêu cầu định dạng và tiêu chuẩn.

- Triển khai mô hình: Đưa mô hình vào môi trường sản xuất bằng cách tích hợp vào hệ thống hoạt động. Điều này có thể liên quan đến việc xây dựng API, giao diện người dùng hoặc tích hợp trực tiếp với hệ thống có sẵn.

- Kiểm tra và theo dõi: Đảm bảo rằng mô hình hoạt động đúng và đáng tin cậy trong môi trường sản xuất. Theo dõi hiệu suất của mô hình và sử dụng các công cụ phù hợp để đảm bảo hoạt động ổn định và khắc phục sự cố nếu có.

- Cập nhật mô hình: Trong trường hợp có dữ liệu mới hoặc cần cải thiện mô hình, tiến hành việc cập nhật mô hình trong môi trường sản xuất. Điều này có thể bao gồm việc thu thập dữ liệu mới, huấn luyện lại mô hình và triển khai phiên bản cập nhật.

Tiếp theo là Ví dụ Continual Learning cho bộ dữ liệu Heart.csv sẽ được sử dụng cho Bài 2, bài báo cáo nhóm.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


# Split the data into an initial training set and a test set
X_train_initial, X_test, y_train_initial, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train an initial model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_initial, y_train_initial)

# Evaluate on the initial test set
y_pred_initial = model.predict(X_test)
accuracy_initial = accuracy_score(y_test, y_pred_initial)
print(f'Initial Test Accuracy: {accuracy_initial}')

test_accuracies = [accuracy_initial]
# Continual Learning: Retrain the model with new data
for i in range(len(X_train_initial), len(X)):
    X_new_data = X.iloc[i:i+1, :]
    y_new_data = y.iloc[i:i+1]

    # Concatenate old and new data
    X_combined = pd.concat([X_train_initial, X_new_data])
    y_combined = pd.concat([y_train_initial, y_new_data])
```

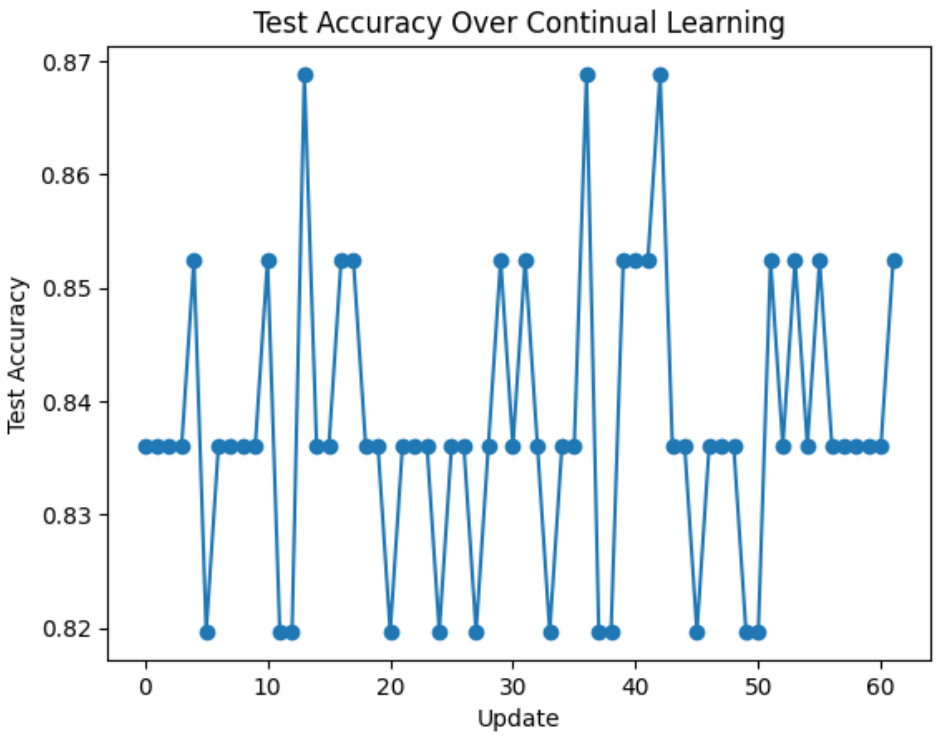
Ví dụ này sử dụng RandomForestClassifier từ scikit-learn. Mô hình ban đầu được đào tạo trên tập dữ liệu ban đầu (X_train_initial, y_train_initial), sau đó mô hình được cập nhật tuần tự với các điểm dữ liệu mới. Độ chính xác của bài kiểm tra được đánh giá sau mỗi lần cập nhật.

Trong ví dụ trên, chúng tôi sử dụng mô hình Logistic Regression để huấn luyện ban đầu và cập nhật mô hình sau đó với dữ liệu mới. Một dữ liệu mới được tạo và nối vào tập huấn luyện, sau đó mô hình được huấn luyện lại. Cuối cùng, chúng tôi đánh giá mô hình đã cập nhật trên tập kiểm tra và in ra độ chính xác.

```python
        # Retrain the model with combined data
        model.fit(X_combined, y_combined)
        y_pred_continual = model.predict(X_test)
        accuracy_continual = accuracy_score(y_test, y_pred_continual)
        print(f'Test Accuracy after update {i}: {accuracy_continual}')
        test_accuracies.append(accuracy_continual)
        print(f'Test Accuracy after update {i}: {accuracy_continual}')
    # Plotting
    plt.plot(range(len(test_accuracies)), test_accuracies, marker='o')
    plt.xlabel('Update')
    plt.ylabel('Test Accuracy')
    plt.title('Test Accuracy Over Continual Learning')
    plt.show()
```

✓ 5.9s

```
Initial Test Accuracy: 0.8360655737704918
Test Accuracy after update 242: 0.8360655737704918
Test Accuracy after update 242: 0.8360655737704918
Test Accuracy after update 243: 0.8360655737704918
Test Accuracy after update 243: 0.8360655737704918
Test Accuracy after update 244: 0.8360655737704918
Test Accuracy after update 244: 0.8360655737704918
Test Accuracy after update 245: 0.8524590163934426
Test Accuracy after update 245: 0.8524590163934426
Test Accuracy after update 246: 0.819672131147541
Test Accuracy after update 246: 0.819672131147541
Test Accuracy after update 247: 0.8360655737704918
Test Accuracy after update 247: 0.8360655737704918
Test Accuracy after update 248: 0.8360655737704918
Test Accuracy after update 248: 0.8360655737704918
Test Accuracy after update 249: 0.8360655737704918
Test Accuracy after update 249: 0.8360655737704918
Test Accuracy after update 250: 0.8360655737704918
Test Accuracy after update 250: 0.8360655737704918
Test Accuracy after update 251: 0.8524590163934426
Test Accuracy after update 251: 0.8524590163934426
Test Accuracy after update 252: 0.819672131147541
Test Accuracy after update 252: 0.819672131147541
Test Accuracy after update 253: 0.819672131147541
Test Accuracy after update 253: 0.819672131147541
...
Test Accuracy after update 301: 0.8360655737704918
Test Accuracy after update 301: 0.8360655737704918
```



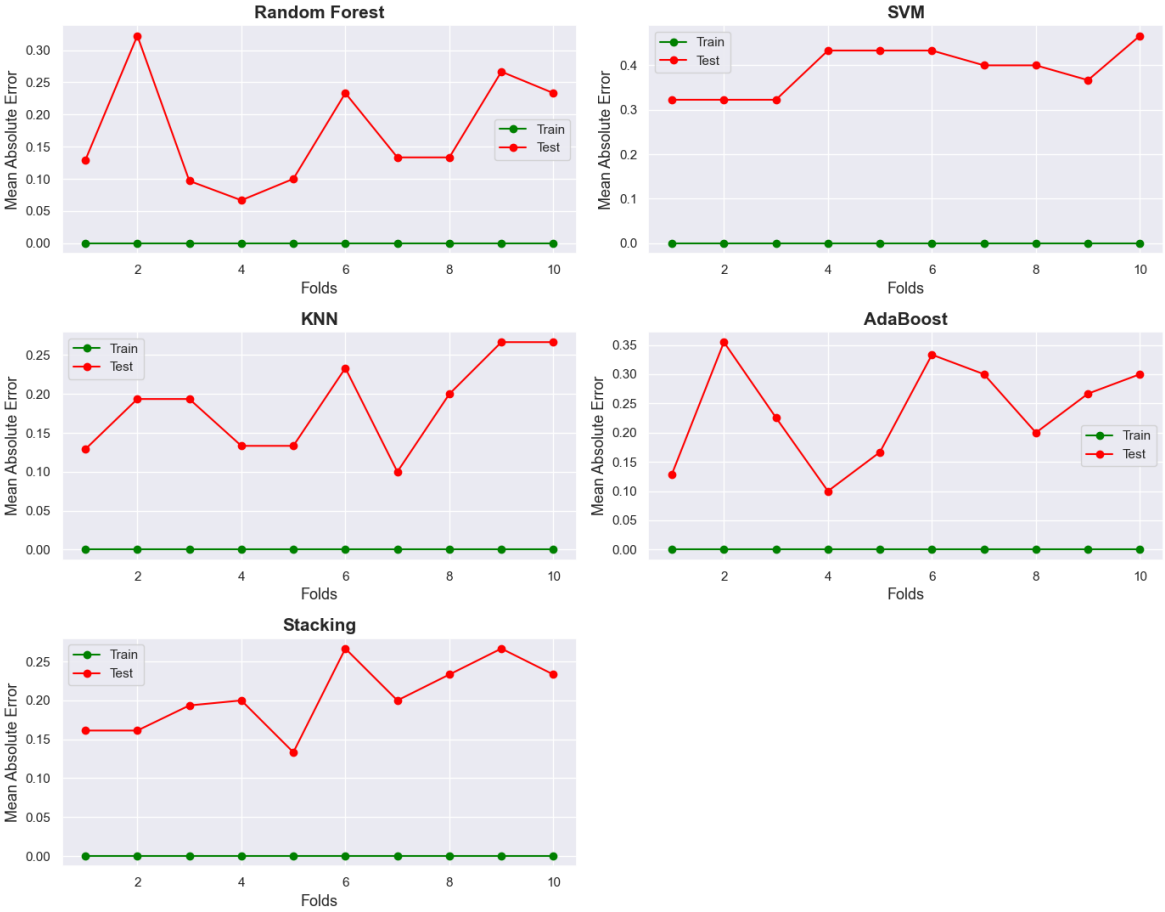
Biểu đồ Test độ chính xác với Continual Learning

Test bộ dữ liệu Heart.csv với 5 mô hình

thực hiện một quá trình huấn luyện và đánh giá nhiều mô hình máy học khác nhau sử dụng phương pháp phân chia dữ liệu K-fold cross-validation.

```
for train_index,test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #random forest
    final_models[0].fit(X_train,y_train)
    test_eval(final_models[0],X_test,y_test,model_names[0])

    #svm
    final_models[1].fit(X_train,y_train)
    test_eval(final_models[1],X_test,y_test,model_names[1])

    #knn
    final_models[2].fit(X_train,y_train)
    test_eval(final_models[2],X_test,y_test,model_names[2])

    #adaboost
    final_models[3].fit(X_train,y_train)
    test_eval(final_models[3],X_test,y_test,model_names[3])

    #stacking
    final_models[4].fit(X_train,y_train)
    test_eval(final_models[4],X_test,y_test,model_names[4])
```

```
sns.set(rc={"figure.figsize":(14, 11)})

folds = range(1,skf.get_n_splits()+1)
for i,models in enumerate(model_compile):
    plt.subplot(3,2,i+1)
    plt.plot(folds,models["Train"],'o-',color='green',label="Train")
    plt.plot(folds,models["Test"],'o-',color='red',label="Test")
    plt.xlabel("Folds",fontsize=13)
    plt.ylabel("Mean Absolute Error",fontsize=13)
    plt.title(model_names[i],fontsize=15,fontweight='bold')
    plt.legend()

plt.tight_layout()
```



Biểu đồ MSE 5 FIRST: thể hiện chỉ số MAE đang rất cao lúc train bằng 5 mô hình **Random forest, SVM, KNN, Adaboost, Stacking** không sử dụng các biện pháp giảm overfitting

DANH MỤC TÀI LIỆU THAM KHẢO

Tiếng việt:

1. Mô hình toán học – Wikipedia tiếng Việt
2. Cây Quyết Định (Decision Tree) - Trí tuệ nhân tạo (trituenhantao.io)

Tiếng anh:

1. Ayush Gupta - September 13th, 2023 - Optimizers in Deep Learning: A Comprehensive Guide (analyticsvidhya.com)
2. GMvandeVen - continual learning - Commits · GMvandeVen/continual-learning · GitHub
3. Continual Learning | Papers With Code
4. Cathrine Jeeva - 4 May 2023 - Model Optimization - Scaler Topics