ADAM: METHOD OPTIMIZATION

Dung P.Viet, **Anh N.Quoc**

Ngày 21 tháng 11 năm 2021

Mục lục

Lời mở đầu

Gradient descent là một trong những thuật toán phổ biến nhất để thực hiện việc tối ưu hóa trong quá trình học máy, và là cách tốt nhất để tối ưu hóa các mạng neural network hiện nay. Có nhiều thuật toán tối ưu khác nhau dựa trên gradient descent đã được đưa ra như Momentum, Adagrad, Adam,. . . Trong báo cáo nghiên cứu này, nhóm học viên sẽ giới thiệu giải thuật, kiến trúc, cách thức hoạt động của các thuật toán trên, đồng thời chỉ ra những thách thức, hạn chế còn tồn tại của chúng. Từ đó, nhóm sẽ đi sâu vào tìm hiểu một trong những thuật toán tối ưu tốt nhất hiện nay là ADAM. Báo cáo sẽ chỉ ra cách thuật toán ADAM giải quyết những hạn chế của những thuật toán đi trước, cũng như những thách thức còn tồn tại của nó.
Nội dung báo cáo gồm ba chương chính: Chương 1 sẽ trình bày những kiến thức cơ sở về gradient descent và những thuật toán tối ưu dựa trên gradient descent. Chương 2 sẽ đi sâu vào tìm hiểu thuật toán ADAM, tập trung chủ yếu vào phân tích thuật toán và đánh giá khả năng hội tụ của ADAM. Trong chương này cũng sẽ giới thiệu một số thuật toán mở rộng dựa trên ADAM. Chương 3 sẽ đi vào phần thực nghiệm, bao gồm các thí nghiệm so sánh hiệu năng giữa ADAM và các thuật toán tối ưu cơ sở, cũng như giữa ADAM và một giải thuật mở rộng của nó là AMSGRAD.

Chương 1

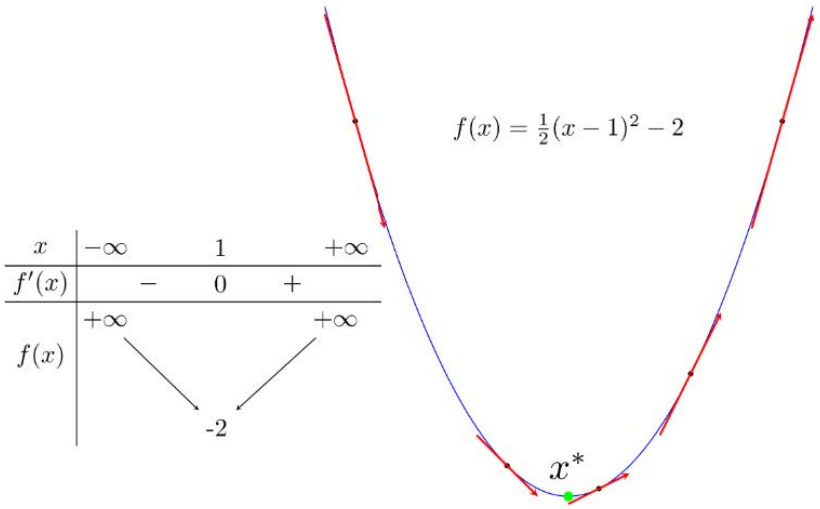
Kiến thức cơ sở

Trong Machine learning nói riêng và Toán tối ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó. Bài toán này có tầm quan trọng trong nhiều lĩnh vực khoa học và kỹ thuật. Ta có thể tìm các điểm cực tiểu hay cực đại của hàm số bằng cách giải phương trình đạo hàm bằng 0. Tuy nhiên, trong đa số các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc có quá nhiều điểm dữ liệu. Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là gần với nghiệm của bài toán, sau đó dùng một phép lặp nào đó để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient descent (viết tắt là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất. Mọi thư viện deep learning hiện nay đều triển khai các thuật toán đa dạng khác nhau dựa trên GD để tối ưu.

1.1 Gradient Descent

1.1.1 Ý tưởng thuật toán

Xét bài toán với hàm 1 biến, xét hàm số $f(x) = \frac{1}{2}(x-1)^2 - 2$ có bảng biến thiên và đồ thị hàm số như hình 1.1. Điểm x^* là điểm cực điểm của hàm số. Tại đ ó $f'(x^*) = 0$. Đường tiếp tuyến với đồ thị hàm số đó tại một điểm bất kỳ có hệ số góc chính bằng đạo hàm của hàm số tại điểm đó. Trong hình 1.1, các điểm bên trái của x^* có đạo hàm âm, các điểm bên phải có đạo hàm dương. Và đối với hàm số này, càng xa về phía trái của x^* thì đạo hàm càng âm, càng xa về phía bên phải thì đạo hàm càng dương.



Hình 1.1: Minh hoạ hàm số $f(x) = \frac{1}{2}(x-1)^2 - 2$

Giả sử x_t là điểm ta tìm được sau vòng lặp thứ t. Ta cần tìm một thuật toán để đưa x_t về càng gần x^* càng tốt.

Từ hình vẽ, chúng ta có quan sát sau:

- Nếu đạo hàm của hàm số tại x_t: $f'(x_t) > 0$ thì x_t nằm về phía bên phải so với x^* (và ngược lại). Để điểm tiếp theo x_{t+1} gần với x^* hơn, chúng ta cần di chuyển x_t về phía bên trái, tức là về phía âm. Hay nói cách khác, chúng ta cần di chuyển ngược dấu với đạo hàm:

$$x_{t+1} = x_t + \Delta$$

Trong đó Δ là một đại lượng ngược dấu với đạo hàm $f'(x_t)$

- x_t càng xa x^* về phía bên phải thì $f'(x_t)$ càng lớn hơn 0 (và ngược lại). Vậy, lượng di chuyển Δ, một cách trực quan, là tỉ lệ thuận với $-f'(x_t)$.

Từ nhận xét phía trên, chúng ta có một cập nhật đơn giản là:

$$x_{t+1} = x_t - \eta f'(\S_t)$$

Trong đó η là một số dương được gọi là learning rate (tốc độ học). Dấu trừ thể hiện việc chúng ta phải đi ngược dấu đạo hàm (Đây cũng chính là lý do phương pháp này được gọi là Gradient Descent - đi ngược đạo hàm).

1.1.2 Batch Gradient Descent

Batch Gradient Descent tức là tính toán gradient cho hàm mục tiêu với tham số cho toàn bộ tập dữ liệu.Ở phần trên, chúng ta đã xét hàm một biến. Tiếp theo chúng ta sẽ xét tổng quát với hàm nhiều biến.
Giả sử ta cần tìm global minimum cho hàm $f(\theta)$ trong đó θ là một vector, thường được dùng để ký hiệu tập hợp các tham số của mô hình cần tối ưu. Đạo hàm của hàm số đó tại một điểm θ bất kỳ được ký hiệu là $\nabla_\theta f(\theta)$. Tương tự như hàm 1 biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán θ_0, sau đó, tại vòng lặp thứ t, quy tắc cập nhật là:

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

Với batch gradient descent, chúng ta cần tính toán gradient cho toàn bộ dataset để biểu diễn một cập nhật, chính vì thế batch gradient descent có thể rất chậm và khó khăn để tính toán cho toàn bộ dataset vì không thể fit hết data vào bộ nhớ. Batch gradient descent cũng không thế cập nhật dưới dạng online learning, tức là cập nhật khi có một mẫu dữ liệu mới đưa vào thì thuật toán phải cập nhật. Nếu làm theo Batch Gradient Descent, tức là tính lại đạo hàm của hàm mục tiêu tại tất cả các điểm dữ liệu, thì thời gian tính toán sẽ rất lâu, và thuật toán của chúng ta coi như là không online nữa do mất quá nhiều thời gian tính toán.
Trên thực tế, có một thuật toán đơn giản hơn và tỏ ra hiệu quả hơn, có tên gọi là Stochastic Gradient Descent (SGD).

1.1.3 Stochastic Gradient Descent

Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mục tiêu dựa trên chỉ một điểm dữ liệu x_i rồi cập nhật θ dựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu, sau đó lặp lại quá trình trên. Thuật toán đơn giản này trên thực tế làm việc rất hiệu quả.

Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với Gradient Descent thông thường thì mỗi epoch ứng với 1 lần cập nhật θ, với SGD thì mỗi epoch ứng với N lần cập nhật θ với N là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu mọt lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). Vì vậy, SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn và các bài toán yêu cầu mô hình thay đổi liên tục, tức là online learning.

Một điểm cần lưu ý là sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các điểm dữ liệu để đảm bảo tính ngẫu nhiên. Việc này ảnh hưởng tới hiệu năng của SGD.

Tóm lại, SGD thực hiện cập nhật tham số cho từng mẫu training $x^{(i)}$ với label $y^{(i)}$ như sau:

$$\theta \equiv \theta - \eta.\nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

Trong khi Batch Gradient Descent cho một đồ thị "mượt" giảm dần, thì SGD giúp chúng ta có thể nhảy đến một tham số tốt hơn. Mặt khác, nó làm phức tạp hoá sự hội tụ chính xác, vì SGD có thể tiếp tục vượt qua điểm cực tiểu. Đồ thị hội tụ của nó không ổn định. Tuy nhiên, các nghiên cứu cho thấy rằng khi chúng ta giảm dần learning rate. SGD cho thấy nó hội tụ nhanh hơn batch gradient descent, gần như chắc chắn hội tụ đến một điểm cục bộ hoặc toàn cục để tối ưu hàm mục tiêu.

1.1.4 Mini-Batch Gradient Descent

Khác với SGD, mini-batch sử dụng mọt số lượng n lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu N rất nhiều). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia dữ liệu thành các mini-batch, mỗi mini-batch có n điểm dữ liệu (trừ mini-batch cuối cùng có thể ít hơn nếu N không chia hết cho n). Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật. Công thức có thể viết dưới dạng:

$$\theta \equiv \theta - \eta.J(\theta; x^{(i:i+n)}, y^{(i:i+n)})$$

Với $x^{(i:i+n)}$ được hiểu là dữ liệu từ thứ i đến thứ i + n 1 (theo ký hiệu của Python). Dữ liệu này sau mỗi epoch khác nhau chúng ta cần được xáo trộn. Các thuật toán khác cho GD như momentum, adagrad, adadelta ... mà chúng ta sẽ tìm hiểu các phần sau cũng có thể áp dụng vào đây.

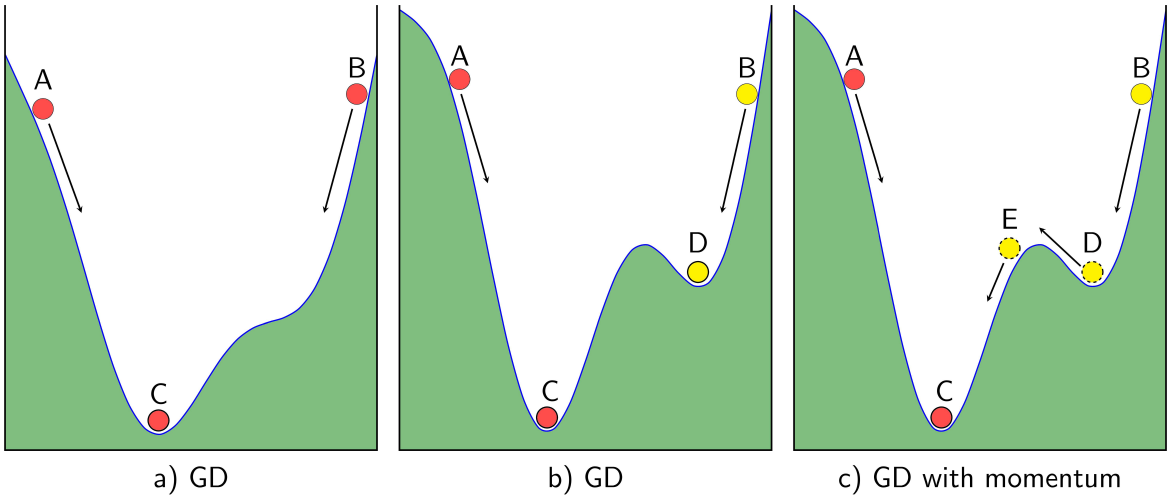
Mini-Batch GD được sử dụng trong hầu hết các thuật toán của Machine Learning, đặc biệt là Deep Learing. Giá trị n thường được lựa chọn khoảng 50 đến 256.

Nhìn chung, gradient descent làm giảm phương sai cập nhật tham số (hội tụ ổn định hơn) so với SGD.

1.2 Các vấn đề của Gradient Descent

Thuật toán mini-batch gradient descent không đảm bảo hội tụ tốt, có một vài vấn đề chúng ta cần giải quyết:

- Chọn 1 learning rate phù hợp: Learning rate quá nhỏ dẫn đến hội tụ chậm và khó khăn, trong khi learning rate quá lớn có thể cản trở sự hội tụ và khiến hàm mất mát dao động xung quanh mức tối thiểu hoặc thậm chí là phân kỳ.
- Xếp lịch cho tỷ lệ học tập có thể điều chỉnh learning rate trong quá trình training, learning rate có thể xác định trước hoặc khi mục tiêu đến một ngưỡng nào đó. Tuy nhiên, cách này thì learning rate phải được xác định lịch biểu từ trước và không thể thích ứng với đặc điểm của mọi bộ dữ liệu.
- Ngoài ra, gradient descent áp dụng một learning rate cố định cho mọi tham số khác nhau, các tham số khác nhau có thể cần những learning rate khác nhau để cập nhật hiệu quả.
- Một khó khăn quan trọng nữa là vượt qua các điểm cực tiểu địa phương, và đặc biệt là các điểm yên ngựa (các điểm này có đạo hàm gần như bằng 0 ở mọi chiều). Một trong số những ý tưởng là thử nhiều lần gradient descent với các điểm khởi tạo ban đầu khác nhau và so sánh các lần, trong nhiều trường hợp thì cách này có thể phát hiện ra điểm cực tiểu địa phương, nhưng không chắc chắn để điểm tìm được là điểm cực tiểu toàn cục của bài toán.



Hình 1.2: So sánh gradient với các hiện tượng vật lý

Để giải quyết các vấn đề này, ở phần tiếp theo, chúng ta sẽ cùng tìm hiểu các thuật toán được cải tiến dựa trên gradient descent như momentum, NAG, AdaGrad...

1.3 Momentum

Thuật toán GD thường được ví với tác dụng của trọng lực lên một hòn bi đặt trên một mặt có dạng như hình một thung lũng như hình 1.2a. Bất kể ta đặt hòn bi ở A hay B thì cuối cùng sẽ lăn xuống vị trí kết thúc C. Tuy nhiên, nếu như bề mặt có hai đấy thung lũng như hình 1.2b thì tuỳ vào việc đặt bi ở A hay B, vị trí cuối cùng của bi sẽ ở C hoặc D. Điểm D được gọi là một điểm cực tiểu địa phương chúng ta không mong muốn, vấn đề này chúng ta đã thảo luận ở phần các vấn đề của gradient descent bên trên.

Nếu suy nghĩ một cách vật lý hơn, vẫn trong hình 1.2b, nếu vận tốc ban đầu của bi khi ở điểm B đủ lớn, khi bi lăn đến điểm D, theo đà, bi có thể tiếp tục di chuyển lên dốc phía bên trái của D. Và nếu giả sử vận tốc ban đầu lớn hơn nữa, bi có thể vượt dốc tới điểm E rồi lăn xuống C như trong hình 1.2c. Đây chính là điều chúng ta mong muốn. Dựa trên hiện tượng này, một thuật toán được ra đời nhằm khắc phục việc nghiệm của GD rơi vào một điểm cực tiểu đại phương không mong muốn. Thuật toán đó có tên là Momentum.

Trong GD, chúng ta cần tính lượng thay đổi ở thời điểm t để cập nhật vị trí mới cho nghiệm (tức hòn bi). Nếu chúng ta coi đại lượng này như vận tốc v_t trong vật lý, vị trí mới của hòn bi sẽ là $\theta_{t+1} = \theta_t - v_t$. Dấu trừ thể hiện việc phải di chuyển ngược với đạo hàm. Công việc của chúng ta bây giờ là tính đại lượng v_t sao cho nó vừa mang thông tin của đạo hàm, vừa mang thông tin của đà (tức vận tốc trước đó v_{t-1}), chúng ta coi vận tốc ban đầu $v_0 = 0$.Một cách đơn giản nhất, ta có thể cộng (có trọng số) hai đại lượng này:

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

Trong đó thường chọn là một giá trị khoảng 0.9, vt1 chính là vận tốc tại thời điểm trước đó, J() chính là gradient (độ dốc) của điểm trước đó. Sau đó vị trí mới của hòn bi được xác định như sau:

$$\theta = \theta - v_t$$

Tóm lại chúng ta có công thức đầy đủ của momentum như sau:

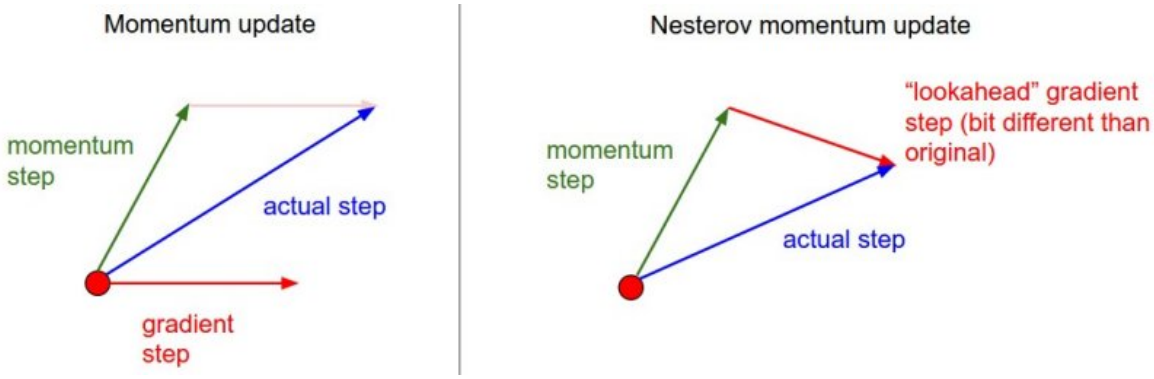
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Về cơ bản, khi sử dụng momentum, giống như chúng ta đẩy một quả bóng xuống thung lũng. Quả bóng tích luỹ vận tốc khi n lăn xuống và ngày càng nhanh hơn trên đường đi, và quả bóng này chịu tác dụng của lực cản như là không khí,.., chính vì thế mà chúng ta có tham số $\gamma < 1$.

1.4 Nesterow accelerated gradient

Momentum giúp hòn bi vượt dốc cực tiểu địa phương, tuy nhiên, có một hạn chế là khi gần tới đích, momentum vẫn mất nhiều thời gian trước khi dừng lại. Điều này chính là do có đà. Nesterow accelerated gradient (viết tắt là NAG) giúp khắc phục điều này để cho thuật toán hội tụ nhanh hơn.

Ý tưởng cơ bản là dự đoán hướng đi trong tương lai, tức nhìn trước một bước. Cụ thể, nếu sử dụng số hạng momentum γv_{t-1} để cập nhật thì ta có thể xấp xỉ được vị trí tiếp theo của hòn bi là $\theta - \gamma v_{t-1}$ (chúng ta không đính kèm phần gradient ở đây vì sẽ sử dụng ở bước cuối cùng). Vậy, thay vì sử dụng gradient của điểm hiện tại, NAG đi dự đoán trước bước tiếp theo.



Theo dõi hình 1.3:

- Với momentum thông thường: lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm hiện tại.
- Với Nesterove momentum: lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm được xấp xỉ là điểm tiếp theo.

Từ trên, ta có công thức cập nhật của NAG như sau:

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

1.5 Adagrad

Như đã thảo luận ở mục các vấn đề của gradient descent, thuật toán gradient descent áp dụng 1 learning rate cho tất cả các tham số. Tuy nhiên, trên thực tế, đôi khi có những tham số ít xuất hiện và cần cập nhật bước sải lớn hơn (high learning rate), các tham số xuất hiện thường xuyên cần cập nhật bước sải nhỏ hơn (low learning rate). Điều này thường xuyên gặp ở nhiều bài toán mà dữ liệu có dạng thưa (sparse). Adagrad ra đời nhằm giải quyết vấn đề này, nó điều chỉnh learning rate theo các tham số. Pennington và đồng nghiệp đã sử dụng Adagrad để train Glove (một mô hình word embedding trong NLP), vì những từ không xuất hiện thường xuyên đòi hỏi cập nhật bước sải lớn hơn rất nhiều so với những từ thường xuyên và đã đạt được hiệu quả tốt hơn so với gradient descent.

Trước đây, chúng ta thực hiện một update cho tất cả các tham số θ, cùng một lúc mỗi tham số θ_i sử dụng learning rate giống nhau η. Adagrad sử dụng learning rate khác nhau cho mỗi tham số θ_i tại mỗi bước t.

Gọi g_t là gradient tại bước t.

$g_{t,i}$ là đạo hàm riêng của hàm mục tiêu theo tham số θ_i tại bước thứ t:

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

Khi đó, GD update cho mỗi tham số θ_i tại bước thứ t trở thành:

$$\theta_{t+1,i} = \theta_{t,i} - \eta . g_{t,i}$$

Trong quy tắc cập nhật của mình, Adagrad sửa đổi sinh learning rate cho mỗi bước t cho mỗi tham số θ_i dựa trên các gradient quá khứ đã được tính toán cho θ_i:

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

Trong đó, $G_t \in R^{dxd}$ là một ma trận đường chéo mà các phần tử trên đường chéo i, i là tổng bình phương của các gradient theo θ_i cho đến bước thứ t, và

là hệ số để tránh chia cho 0 (thường là 10^{-8}). Theo thử nghiệm, nếu không có căn bậc 2 dưới mẫu, thuật toán lại hoạt động tệ hơn nhiều.

Như vậy, G_t gồm tổng bình phương các gradient quá khứ của tất cả các tham số θ dọc theo đường chéo, chúng ta có thể viết tổng quát lại công thức cập nhật cho Adagrad như sau:

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

Thông thường, $\eta = 0.01$. Rõ ràng, lợi ích của Adagrad là ước lượng để điều chỉnh learning rate cho từng tham số. Tuy nhiên, điểm yếu chính của Adagrad là sự tích luỹ gradient ở mẫu số lớn dần lên trong quá trình training. Điều này làm cho learning rate thu hẹp lại và dần trở lên vô cùng nhỏ, tại thời điểm đó, thuật toán không còn có thể học được nữa. Các thuật toán sau này nhằm giải quyết lỗ hổng này.

1.6 Adadelta

Adadelta là một mở rộng của Adagrad nhằm tìm cách giảm bớt nhược điểm của nó, đó là learning rate giảm dần. Thay vì tích luỹ bình phương gradient quá khứ, Adadelta hạn chế tích luỹ quá khứ đến một kích thước cố định w.

Để làm được điều trên, nó đưa ra khái niệm trung bình trượt (running average). Trung bình trượt $E[g^2]_t$ tại thời điểm t chỉ phụ thuộc vào trung bình trượt tại thời điểm trước và gradient hiện tại:

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

Trong đó, γ thường được đặt là 0.9.

Gọi $\Delta\theta_t$ là đại lượng thay đổi cho tham số tại bước thứ t.

Khi đó, với GD, ta có thể viết lại công thức như sau:

$$\Delta\theta_t = -\eta . g_t$$

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

Tương tự, với Adagrad thì ta có:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \varepsilon}} \odot g_t$$

Bây giờ chúng ta thay ma trận đường chéo G_t bằng $E[g^2]_t$:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

Mẫu số chính là root mean squared error (RMS) của gradient, chúng ta có thể viết ngắn gọn lại:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

Ý tưởng tiếp theo là thuật toán sử dụng ý tưởng correct các đơn vị với nhau, xuất phát từ thuật toán Newton, ta có:

$$\Delta\theta_t = H_t^{-1} g_t$$

Trong đó, H_t^{-1} là nghịch đảo của ma trận Hessian (đạo hàm bậc 2 của hàm mất mát). Từ đó, ta có:

$$\Delta\theta \propto H^{-1} g \propto \frac{\partial f/\partial\theta}{\partial^2 f/\partial\theta^2}$$
$$\Rightarrow \Delta\theta = \frac{\partial f/\partial\theta}{\partial^2 f/\partial\theta^2} \Rightarrow \frac{1}{\frac{\partial^2 f}{\partial\theta^2}} = \frac{\Delta\theta}{\partial f}$$

Với f là hàm mục tiêu.
Từ trên, vì gradient của f trước đó đã được biểu diễn trong mẫu số xấp xỉ là $RMS[g]_t$, ta coi $\Delta\theta$ như là tham số, vì $\Delta\theta_{t-1}$ là chưa biết, chúng ta sẽ xấp xỉ nó bằng $RMS[\theta]_{t-1}$, thay thế learning rate trong biểu thức ban đầu thành $RMS[\Delta\theta]_{t-1}$, ta có quy tắc cập nhật Adadelta như sau:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Khởi tạo $\Delta\theta_0 = 0$, với Adadelta, chúng ta không cần thiết lập learning rate mặc định, vì nó đã bị loại bỏ khỏi quy tắc cập nhật.

1.7 RMSProp

RMSProp là phương pháp điều chỉnh learning rate được đưa ra bởi Geoff Hinton. RMSProp và Adadelta phát triển độc lập cùng lúc xuất phát từ nhu cầu giải

quyết triệt để learning rate giảm dần của Adagrad. RMSProp có công thức cập nhật như sau:

$$E[g^2]_t = 0.9[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

RMSProp cũng chia learning rate cho $E[g^2]_t$. Hinton đề nghị $\gamma = 0.9$ và giá trị mặc định tốt cho $\eta = 0.001$.

Chương 2

Thuật toán Adam

2.1 Thuật toán

Algorithm 1: Adam, Ký hiệu g_t^2 là bình phương của phép nhân element-wise $g1 \odot g2$,các tham số khuyến nghị là $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ và $\varepsilon = 10^{-8}$. Tất cả các toán tử trên vector là phép element-wise (tức là thực hiện tương ứng từng phần tử)

Require: α: Stepsize;
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates;
Require: $f(\theta)$: Stochastic objective function with parameters θ;
Require: θ_0: Initial parameter vector;
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector);
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector);
$t \leftarrow 0$ (Initialize timestep);
while θ_t not converged **do**:
 $t \leftarrow t + 1$;
 $g_t \leftarrow \triangle_\theta(\theta_{t-1})$;
 $m_t \leftarrow \beta_1.m_{t-1} + (1 - \beta_1).g_t$ (Update biased fincecond rauate) ;
 $v_t \leftarrow \beta_2.v_{t-1} + (1 - \beta_2).g_t^2$ (Compute biasceted finate) ;
 $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias corrected first moment estimate) ;
 $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias corrected second raw moment estimate) ;
 $\theta_t \leftarrow \theta_{t-1} - \alpha.\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters) .
end

Result: Trả về tham số θ_t

2.2 Phân tích và đánh giá thuật toán

Adam, (Adaptive moment Estimation), là một phương pháp ước lượng learning rate cho mỗi tham số. Adam được coi như là sự kết hợp của AdaDelta hay RMSProp và momentum. Trong khi momentum có thể xem như một quả bóng đang chạy xuống dốc, Adam lại giống như một quả bóng nặng với ma sát. Adam sử dụng bình phương gradient để chia tỷ lệ learning rate như RMSProp và tận dụng "đà" giống như momentum

Công thức cập nhật đầy đủ của Adam:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_{t+\varepsilon}}} \end{aligned}$$

Để tốt hơn về chi phí tính toán, 3 dòng cuối cùng ta có thể viết lại như sau:

$$\begin{aligned} \eta_t &= \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \\ \theta_t &= \theta_{t-1} - \eta_t \frac{m_t}{\sqrt{v_t}+\varepsilon} \end{aligned}$$

2.2.1 Bias Correction

Adam tính toán moment cấp 1 và moment cấp 2 của gradient để ước lượng learning rate cho các tham số.

Định nghĩa 1 (Khái niệm moment). *Moment cấp k đối với a của biến ngẫu nhiên X là một số xác định như sau:*

$$v_k(a) = E[(X-a)^k]$$

Nếu a = 0, ta ký hiệu $v_k(a) = E[(X-a)^k]$ và gọi nó là moment gốc cấp k. Rõ ràng kỳ vọng chính là moment gốc cấp 1 EX = v1. Nếu a = EX, ta ký hiệu $\mu_k = v_k(EX) = E[(X-EX)^k]$ và gọi nó là moment trung tâm cấp k; cũng rõ ràng phương sai là moment trung tâm cấp 2 VX = μ_2

Người ta còn dùng moment để đặc trưng cho hình dạng của mật độ phân phối.

Như đã thấy ở thuật toán, Adam sử dụng thuật ngữ khởi tạo bias correction. Chúng ta sẽ làm rõ điều này cho ước lượng moment cấp 2; nguồn gốc ước lượng moment cấp 1 là tương tự.

Ta thấy rằng, gradient của hàm mục tiêu có thể xem như là một biến ngẫu

nhiên, vì nó thường được đánh giá trên một mini-batch dữ liệu. Moment cấp 1 là kỳ vọng, moment cấp 2 là uncentered variance (Có nghĩa là moment gốc cấp 2 - chúng ta không trừ đi 11 (2.1) giá trị trung bình khi tính toán phương sai). Để ước lượng các moment, Adam tận dụng exponentially moving average của gradient và bình phương gradient, với decay rate là β_1 và β_2.

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

Với mt và vt được gọi là moving averages, g là gradient trên mini-batch hiện tại. Để xem các giá trị này tương quan với moment như thế nào, chúng ta hãy xem xét giá trị kỳ vọng của các moving averages. Vì m và v là các ước lượng của moment cấp 1 và moment cấp 2, chúng ta muốn có tính chất sau:

$$E[m_t] = E[g_t]$$
$$E[v_t] = E[g_t^2]$$

Ta khai triển vt như sau (tương tự với mt):
$v_0 = 0$
$v_1 = \beta_2 v_0 + (1 - \beta_2) g_1^2 = (1 - \beta_2) g_1^2$
$v_2 = \beta_2 v_1 + (1 - \beta_2) g_2^2 = \beta_2 (1 - \beta_2) g_1^2 + (1 - \beta_2) g_2^2$
$\dots$
$\mathrm{v}_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} . g_i^2$
Do $\beta_2 \in [0, 1)$, nên có thể thấy rằng, exponential moving average đánh trọng số cho các gradient, càng các gradient đầu tiên thì càng đóng góp ít vào giá trị tổng thể, vì chúng được nhân với β nhỏ hơn. Bây giờ chúng ta sẽ xem xét giá trị kỳ vọng của vt, để xem nó liên quan thế nào đến giá trị thực moment cấp 2 là $E[g_t^2]$. Ta có:

$$E[v_t] = E[(1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} . g_i^2]$$
$$= E[g_t^2] . (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} + \zeta$$
$$= E[g_t^2] . (1 - \beta_2^t) + \zeta$$

Ở khai triển trên, chúng ta đã sử dụng xấp xỉ gi thành gt và đưa nó ra khỏi tổng. Vì sự xấp xỉ này, nên ta cần cộng thêm một đại lượng ζ trong công thức để đại diện cho sai lệch do xấp xỉ. Tiếp theo, chúng ta cần điều chỉnh giá trị ước lượng của v_t để cho kỳ vọng của nó xấp xỉ moment cấp 2, sở dĩ có sự sai lệch đại lượng $(1 - \beta_2^t)$ là do ta đã khởi tạo $v_0 = 0$. Từ đó ta có công thức hiệu chỉnh cho v như sau:

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

Tương tự, thực hiện các bước trên với mt. Cuối cùng, chúng ta có công thức cập nhật tham số cho Adam như sau:

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

2.2.2 Các tính chất về bước sải của Adam

Cho $\varepsilon = 0$. Ta gọi $\Delta_t = \eta.\hat{m}_t/\sqrt{\hat{v}_t}$ là bước sải tại vòng lặp thứ t. Ta có các tính chất quan trọng sau của bước sải:

- **Độ lớn tối đa của bước xải xấp xỉ η**
- **Độ lớn bước xải là bất biến với độ lớn của gradient**

Chứng minh. Thật vậy, thay đổi giá trị của g với hệ số c sẽ tương ứng thay đổi tỉ lệ của $\hat{m}_t$ với hệ số c và $\hat{v}_t$ với hệ số c^2. Từ đó ta có:

$$\frac{c.\hat{m}_t}{\sqrt{c^2.\hat{v}_t}} = \frac{\hat{m}_t}{\hat{v}_t}$$

2.3 Các mở rộng của thuật toán Adam

2.3.1 AdaMax

Quy tắc cập nhật Adam cho từng trọng số riêng là tỉ lệ nghịch với chuẩn L^2 của gradient quá khứ và hiện tại. Chúng ta có thể khái quát hoá chuẩn L^2 thành chuẩn L^p cho quy tắc cập nhật của Adam. Các biến thể như vậy không ổn định cho p lớn. Tuy nhiên, trong trường hợp đặc biệt khi $p \to \infty$ thì ta lại thu được một thuật toán ổn định đáng ngạc nhiên. Trong chuẩn L^p, tại bước thứ t tỉ lệ nghịch với $v_t^{1/p}$, với:

$$v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p)|g_t|^p = (1 - \beta_2^p) \sum_{i=1}^{t} \beta_2^{p(t-i)}.|g_i|^p$$

Algorithm 2: Adamax, Ký hiệu g_t^2 là bình phương của phép nhân element-wise $g1 \odot g2$,các tham số khuyến nghị là $\alpha = 0.002$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. Tất cả các toán tử trên vector là phép element-wise)

Require: α: Stepsize;
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates;
Require: $f(\theta)$: Stochastic objective function with parameters θ;
Require: θ_0: Initial parameter vector;
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector);
$v_0 \leftarrow 0$ (Initialize the exponentially weighted infinit norm);
$t \leftarrow 0$ (Initialize timestep);
while θ_t not converged **do**:
$t \leftarrow t + 1$;
$g_t \leftarrow \triangle_\theta(\theta_{t-1})$;
$m_t \leftarrow \beta_1.m_{t-1} + (1 - \beta_1).g_t$ (Update biased first moment estimate) ;
$\mathrm{u}_t \leftarrow max\big(\beta_2.u_{t-1}, |g_t|\big)$(Update the exponentially weighted infinity norm)
$\theta_t \leftarrow \theta_{t-1} - (\alpha/(1 - \beta_1^t))\, .m_t/u_t$(Update parameters)
end
Result: Trả về tham số θ_t

Lưu ý rằng thuật ngữ phân rã ở đây được tham số hoá là β_2^p thay vì β_2. Bây giờ, xét $p \longrightarrow \infty$ và định nghĩa $\mathcal{U}_t \longrightarrow lim_{p\to\infty}\big(\upsilon_t\big)^{1/p}$, thì:

$$\begin{aligned}
&\mathcal{U}_t \longrightarrow lim_{p\to\infty}\big(\upsilon_t\big)^{1/p} = \lim_{p \to \infty} \left((1 - \beta_2^p) \sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p\right)^{1/p} \\
&= \lim_{p\to\infty} (1 - \beta_2^p)^{1/p} \left(\sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p\right)^{1/p} \\
&= \lim_{p\to\infty} \left(\sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p\right)^{1/p} \\
&= \max(\beta_2^{t-1}|g_1|, \beta_2^{t-2}|g_2|, ..., \beta_2|g_{t-1}|, \, |g_t|)
\end{aligned}$$

Ta có công thức đệ quy đơn giản tương ứng là:

$$u_t = max(\beta_2.u_{t-1}, |g_t|)$$

2.3.2 Nadam

Adam có thể được xem như là sự kết hợp của RMSprop và momentum vì nó sử dụng các trung bình trượt theo cấp số nhân (exponential moving averages) của bình phương gradient và gradient. Mặt khác, ta đã biết Nesterov accelerated gradient (NAG) giúp cho thuật toán hội tụ nhanh hơn so với momentum thông thường. Nadam (Nesterov- accelerated Adaptive Moment Estimation) là một thuật toán kết hợp Adam và NAG. Để phát triển Nadam, ta phải điều chỉnh giá trị momentum mt trong Adam. Đầu tiên, ta xem lại công thức cập nhật

momentum:

$$g_t = \nabla\theta_t J(\theta_t)$$
$$p_t = \gamma m_{t-1} + \eta g_t$$
$$\theta_{t+1} = \theta_t - m_t$$

Trong đó, $\nabla\theta_t \mathcal{J}(\theta_t)$ là gradient của điểm trước đó, m_t là momentum của điểm trước đó, γ thường được chọn với giá trị 0.9, η là learning rate.
Mở rộng phương trình trên ta được:

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t)$$

Có thể thấy với momentum thông thường, lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm hiện tại.
Với Nesterove momentum, lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm được xấp xỉ là điểm tiếp theo. Ta có công thức của NAG:

$$\mathrm{g}_t = \nabla\theta_t J(\theta_t)$$
$$\mathrm{m}_t = \gamma m_{t-1} \ + \eta g_t$$
$$\theta_{t+1} = \theta_t - (\gamma\eta m_t + \eta g_t)$$

Có thể thấy trong công thức trên, thay vì sử dụng vector momentum mt1 như trong phương pháp momentum, giờ ta sử dụng vector momentum mt để đi trước một bước. Do đó, để thêm Nesterov momentum vào Adam, ta thay các vector momentum quá khứ với vector momentum hiện tại. Đầu tiên, ta xem lại công thức cập nhật của Adam:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$m_t = \frac{m_t}{1-\beta_1^t} \qquad\qquad \eta$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

Mở rộng phương trình trên theo $\hat{m}_t$ và m_t ta có:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\frac{\beta_1 m_{t-1}}{1-\beta_1^t} + \frac{(1-\beta_1)\, g_t}{1-\beta_1^t}\right)$$

Ở đây $\frac{m_t - 1}{1-\beta_1^t}$ là chỉ số bias-corrected của vector momentum của điểm dữ liệu trước. Do đó ta có thể thay nó với $\hat{m}_{t-1}$:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\beta_1 \hat{m}_{t-1} + \frac{(1-\beta_1)\, g_t}{1-\beta_1^t}\right)$$

Bây giờ, chúng ta có thể thêm Nesterov momentum bằng cách thay chỉ số bias-corrected của vector momentum của điểm dữ liệu trước ^mt1 bằng chỉ số bias-corrected của vector momentum hiện tại $\hat{m}_t$. Từ đó, ta có công thức cập nhật của Nadam:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\beta_1 \hat{m}_t + \frac{(1-\beta_1)\, g_t}{1-\beta_1^t}\right)$$

2.3.3 AMSGrad

Như đã thảo luận bên trên, trong một số trường hợp, các phương pháp ước lượng learning rate thích ứng không hội tụ đến giải pháp tối ưu, thậm chí kết quả còn kém hơn SGD với momentum. Trong bài báo của mình, Reddi và các cộng sự đã chỉ ra exponential moving average (EMA) của các bình phương gradient quá khứ là nguyên nhân gây ra hiện tượng này. Ban đầu, EMA được đề xuất để giải quyết vấn đề learning rate trở nên quá nhỏ trong quá trình học của thuật toán Adagrad. Tuy nhiên, các gradient với bộ nhớ ngắn hạn này lại trở thành nhược điểm trong những trường hợp khác.
Trong các cài đặt mà Adam không hội tụ đến kết quả tối ưu, người ta thấy rằng một số mini-batch dữ liệu cung cấp các gradient lớn và chứa nhiều thông tin, nhưng tần suất các mini-batch này rất ít và EMA nhanh chóng giảm bớt ảnh hưởng của chúng đối với kết quả cuối cùng, dẫn đến hội tụ kém.
Để giải quyết vấn đề này, thuật toán AMSGrad được đề xuất. Thuật toán sử dụng giá trị lớn nhất của các bình phương gradient trong quá khứ, thay vì EMA để cập nhật tham số vt được định nghĩa tương tự như trong thuật toán Adam:

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

Khi tính $\hat{v_t}$, thay vì sử dụng v_t, ta sử dụng $\hat{v}_{t-1}$ nếu nó lớn hơn v_t:

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

Bằng cách này, AMSGrad cho kết quả là step size không tăng, từ đó tránh kết quả tồi như thuật toán Adam. Để đơn giản hóa, tác giả bài báo cũng loại bỏ các bước hiệu chỉnh bias trong thuật toán Adam. Công thức đầy đủ của thuật toán AMSGrad như sau:

$$\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\
\hat{v}_t &= max(\hat{v}_{t-1,} v_t) \\
\hat{\theta}_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} m_t
\end{aligned}$$

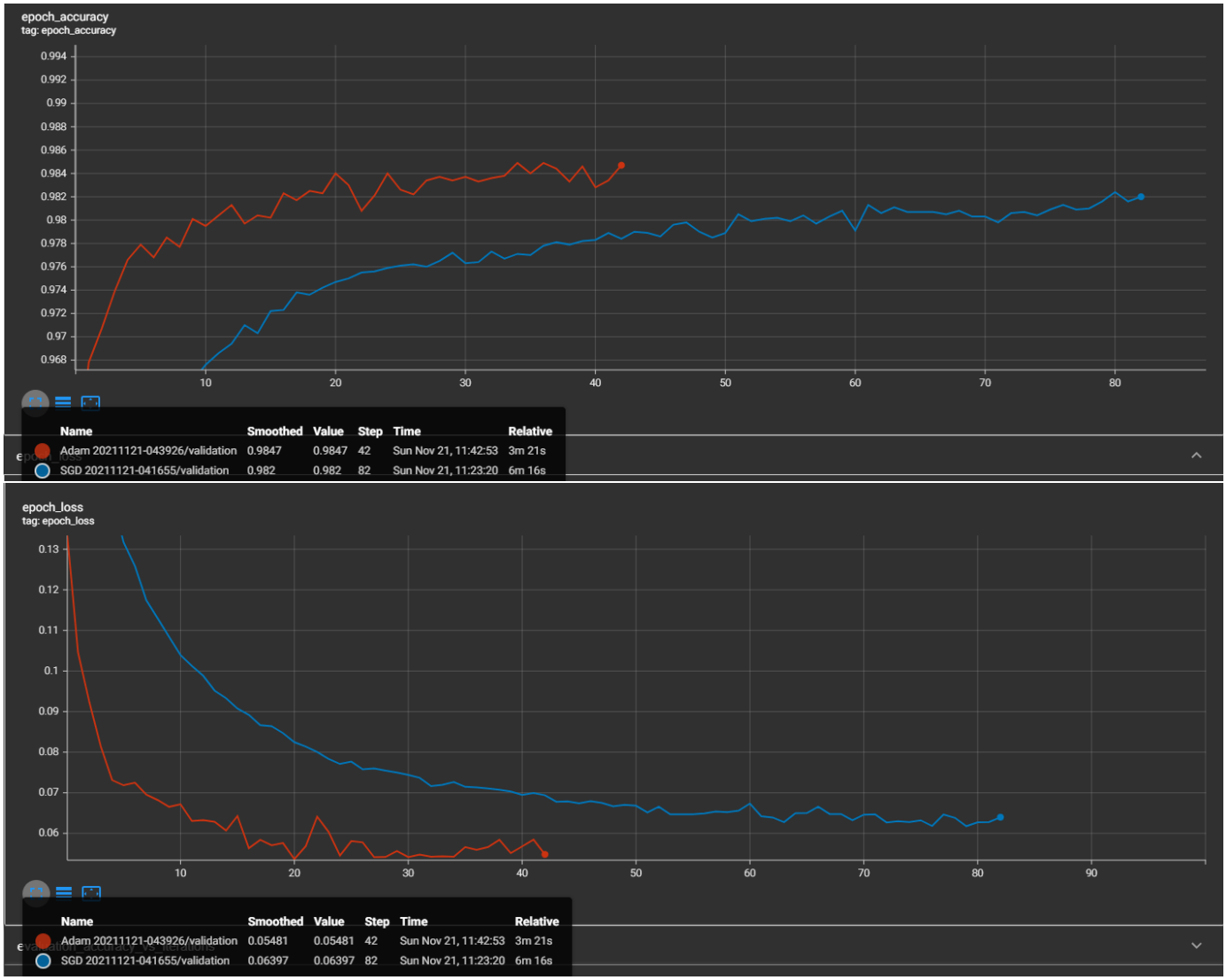
Trong các datasets nhỏ và CIFAR-10, AMSGrad cho kết quả tốt hơn Adam, nhưng trong một số thí nghiệm khác, kết quả của nó lại tương đương, thậm chí kém hơn so với Adam. Vì vậy, việc áp dụng và so sánh hiệu năng AMSGrad so với Adam vẫn cần được tìm hiểu và xem xét nhiều hơn nữa.

Chương 3

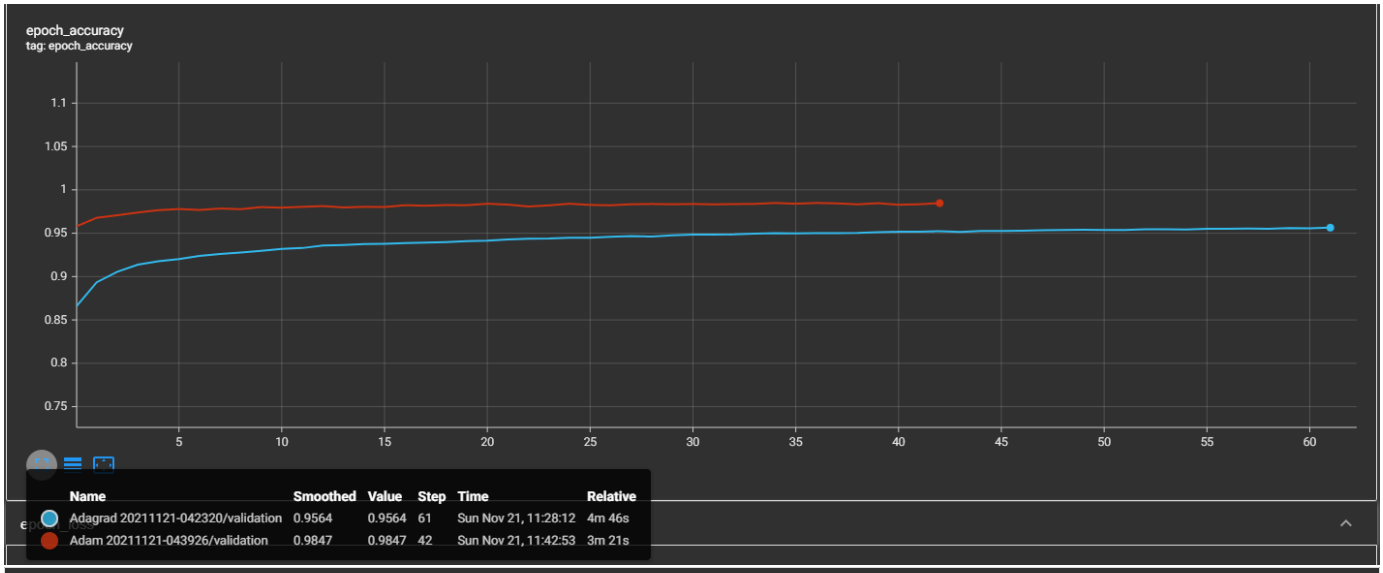
Thực nghiệm với thuật toán Adam

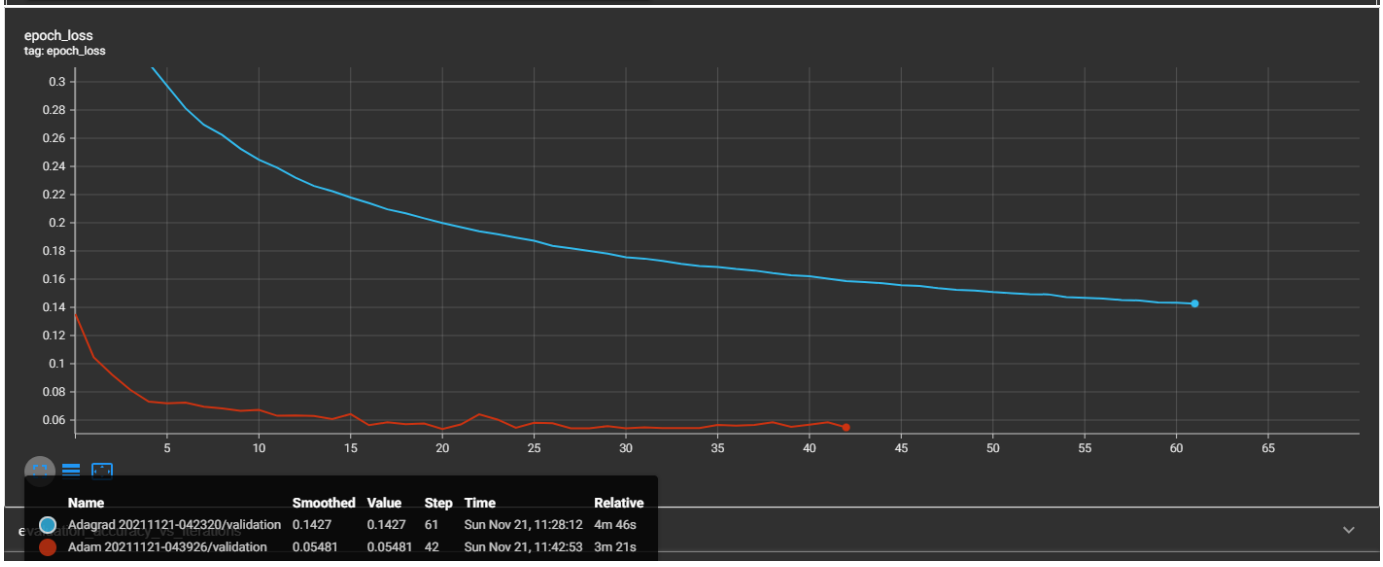
- Bài toán thực hiện là nhận dạng chữ viết tay dựa trên bộ dữ liệu Mnist bao gồm 60000 ảnh trong tập train và 10000 ảnh trong tập test. Bài toán sẽ phân loại các vector hình ảnh 28 * 28 chiều vào 10 lớp label khác nhau.
- Thí nghiệm để so sánh độ hiệu quả của thuật toán Adam với 6 thuật toán tối ưu khác là SGD, Adagrad, Adadelta,RMSprop, Adamax, Nadam.
- Bài toán thực nghiệm trên mạng nơ-ron gồm 3 lớp, với batch size = 64, epochs = 50, toàn bộ được đánh giá trên tập validation.
- Hyper parameters sử dụng trong các thuật toán:
  learning rate=0.001,
  $beta_1$=0.9,
  $beta_2$=0.999,
  $\varepsilon = 10^{-7}$,
  rho=0.9,
  momentum=0.0
- Dưới đây là so sánh cụ thể với từng thuật toán:

3.1 Adam vs SGD

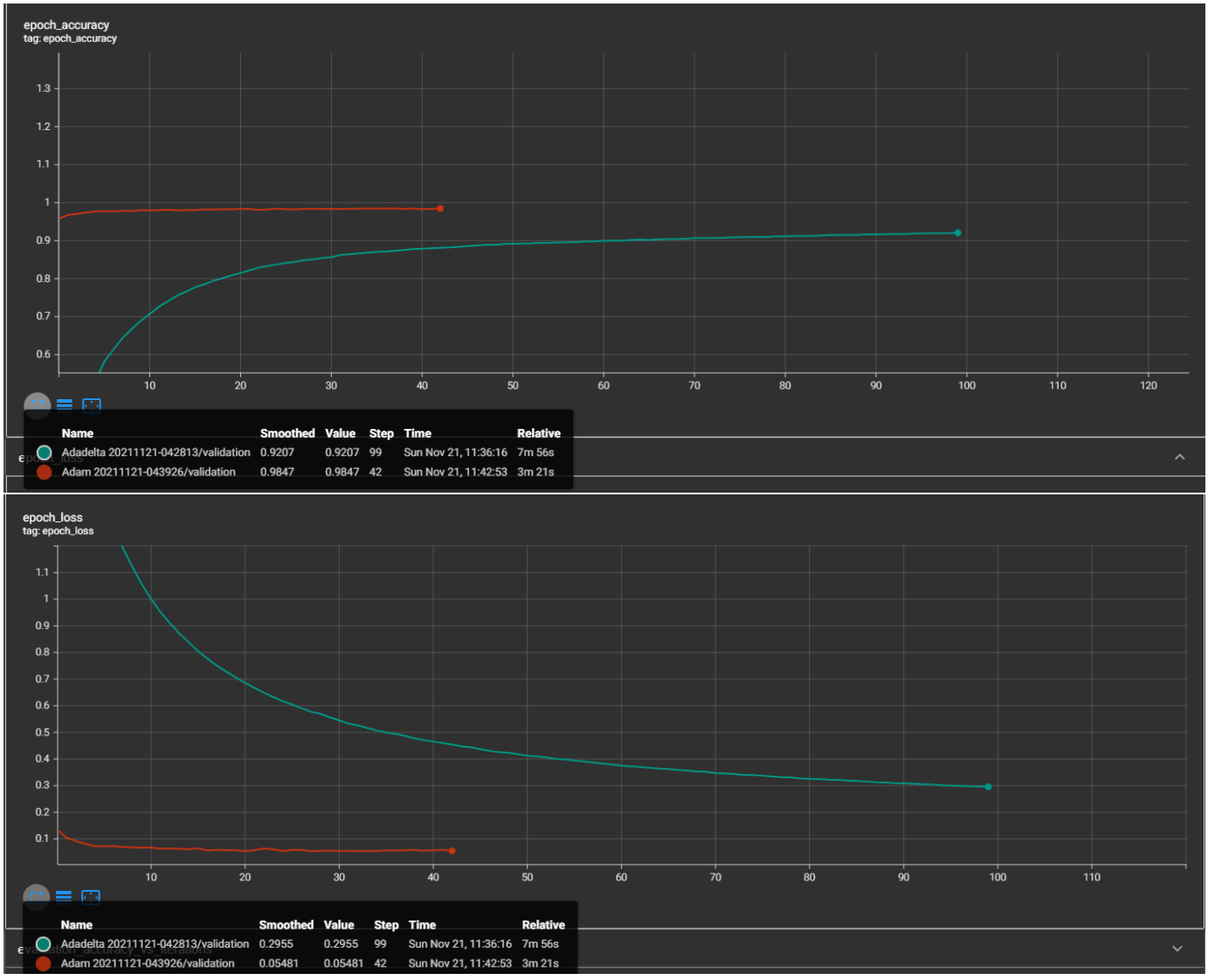


3.2 Adam vs Adagrad



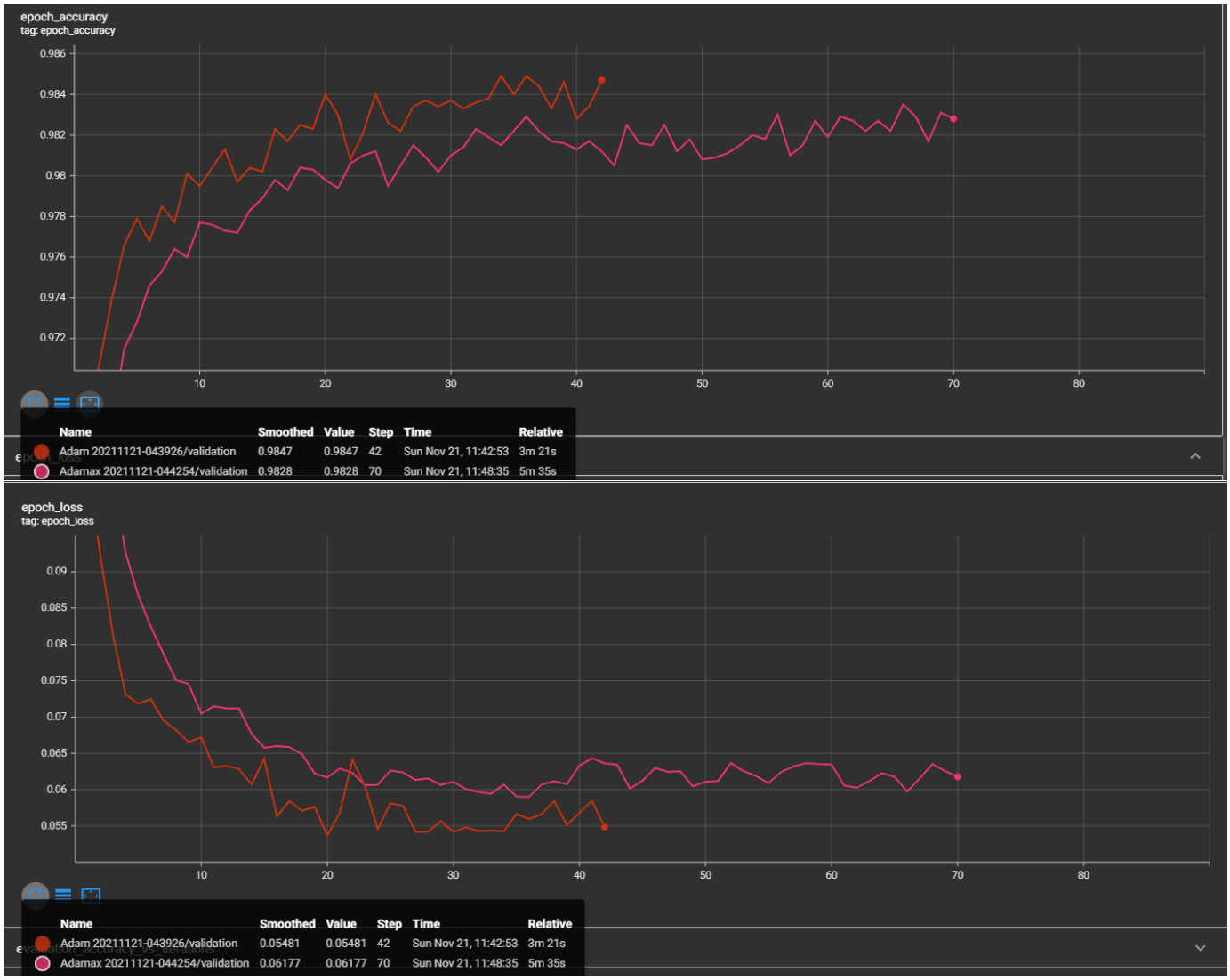


3.3 Adam vs Adadelta

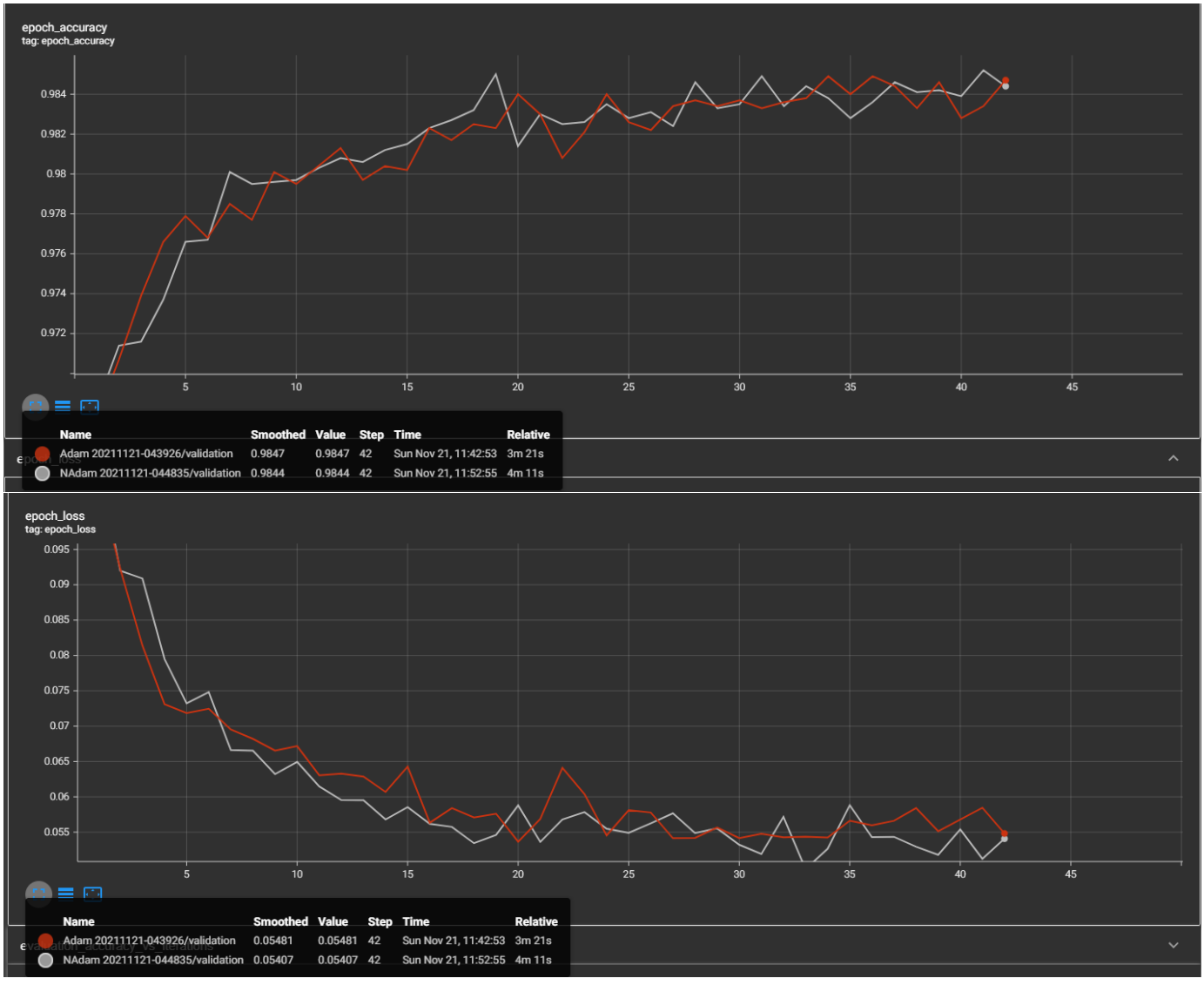


3.4 Adam vs RMSprop

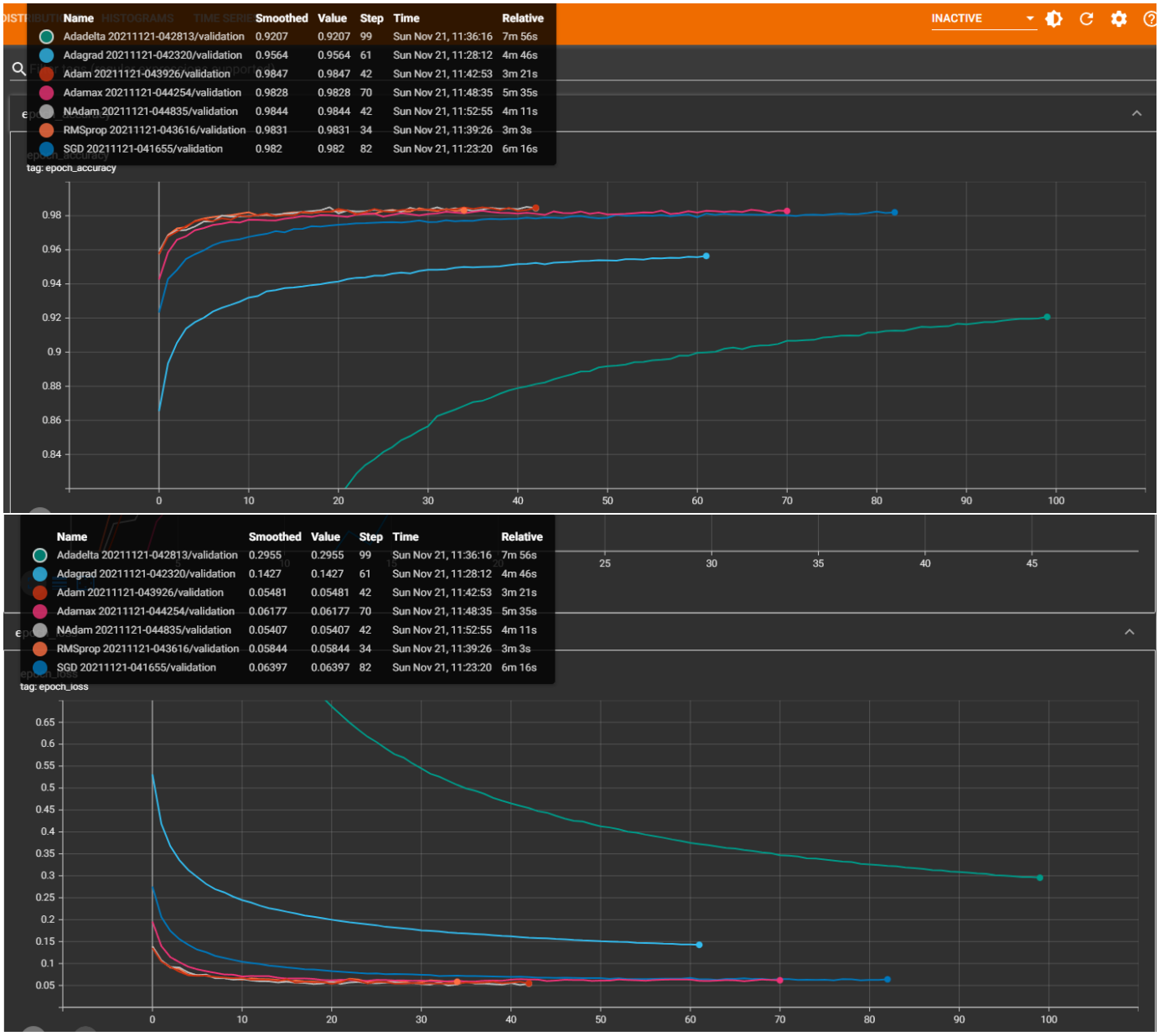
3.5 Adam vs Adamax



3.6 Adam vs Nadam



3.7 Tổng quan



- Từ đồ thị ta thấy, thuật toán Adam hội tụ khá nhanh (epochs 42) và độ hiệu quả của thuật toán khá tốt (accuaracy = 0.9847) so với các thuật

toán khác.

- Đối với thuật toán SGD, Adagrad và Adadelta, sự hội tụ của các thuật toán khá chậm, đặc biệt với Adadelta, độ chính xác của thuật toán rất thấp.
- Thuật toán Adamax tuy hiệu quả cao nhưng thời gian hội tụ khá chậ.m
- Bên cạnh Adam còn RMSprop và NAdam cũng cho ra kết quả rất tốt đối với bài toán này.

Kết luận

Có thể thấy rằng nhiều thuật toán tối ưu hóa dựa trên gradient descent đã được đưa ra, và những thuật toán sau luôn được bổ sung hoàn thiện để giải quyết những tồn tại của thuật toán trước. RMSprop là một mở rộng của Adagrad với khả năng giải quyết vấn đề learning rate trở nên quá nhỏ, tương tự như Adadelta. Sau đó, thuật toán ADAM bổ sung thêm bias-correction và momentum vào RMSprop để mang lại hiệu năng tốt hơn, nó sử dụng các cập nhật gradient tỷ lệ với căn bậc hai của bình phương trung bình trượt các gradient trong quá khứ.Vì vậy, ADAM rất thông dụng trong thực tế, về mặt lý thuyết, cũng như trong một số thực nghiệm mà nhóm đã thực hiện.

Tài liệu tham khảo

[1] Diedelrik P.Kingma, Jimmy Lei Ba, *Adam: A method for stochastic optimization*, ICLR 2015.

[2] Vũ Hữu Tiệp, *Machine learning cơ bản*, Nhà xuất bản khoa học và kỹ thuật, 2018.

[3] Sebastian ruder, *An overview of gradient descent optimization algorithms*, https://ruder.io/optimizing-gradient-descent/index.html

[4] Hassan Dbouk, *On the convergence of sgd, adam and Amsgrad*, ECE 543 project report