TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO MÔN HỌC
HỌC MÁY VỚI DỮ LIỆU LỚN

PHƯƠNG PHÁP TỐI ƯU ADAM

Giảng viên hướng dẫn: **PGS TS. Thân Quang Khoát**

Sinh viên thực hiện: **Nguyễn Đức Thắng - MSSV: 20166769**

Lương Thành Long - MSSV: 20155970

Phạm Xuân Nam Chính - MSHV: CB190194

HÀ NỘI, 4/2020

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

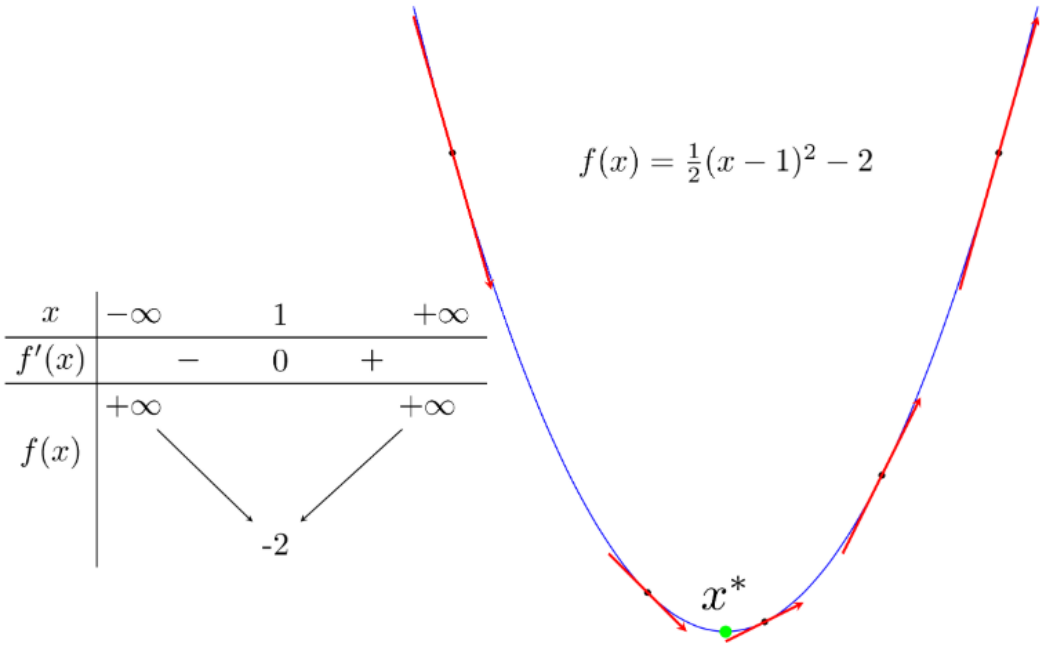
1.1.1 Ý tưởng thuật toán

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 đó $f'(x^*) = 0$. 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.

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



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

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:

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

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 θ 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:

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

. 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:

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

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.

1.1.5 Các vấn đề của Gradient Descent

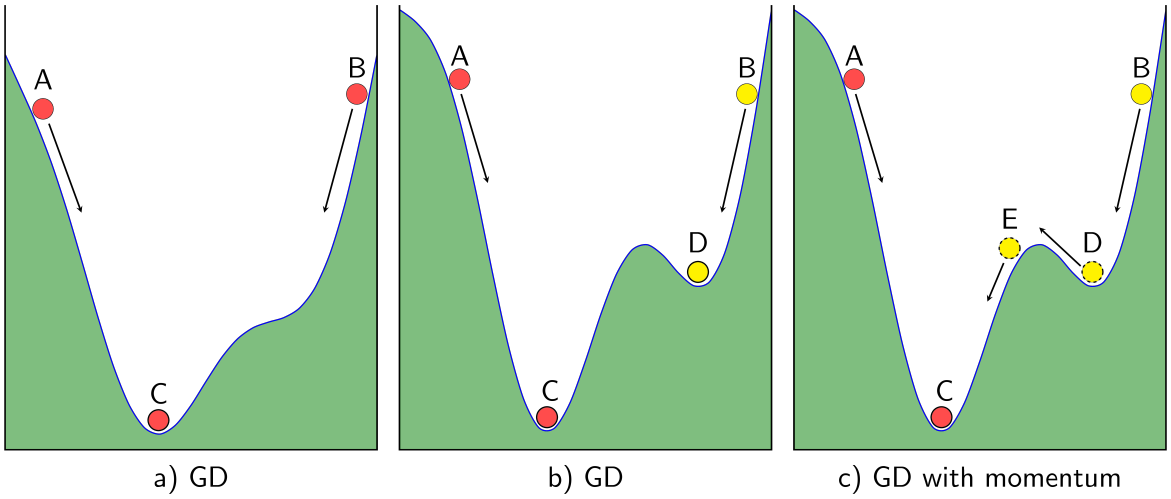
Thuật toán mini-batch gradient descent không đảm bảo hội tụ tốt, có một vài vấn đề chúng ta cần giải quyết:

- 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

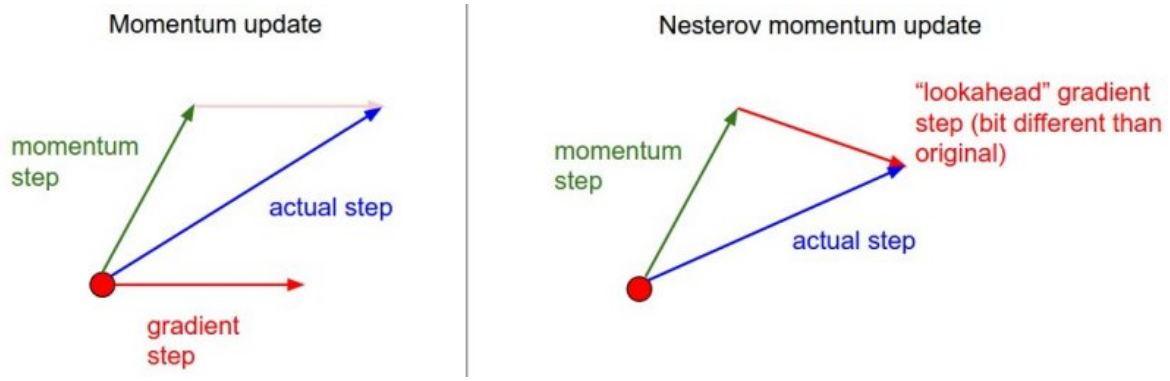
Để giải quyết các vấn đề này, ở phần tiếp theo, chúng ta sẽ cùng tìm hiểu các thuật toán được cải tiến dựa trên gradient descent như momentum, NAG, AdaGrad...

1.2 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 vậy 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



Hình 1.3: Ý tưởng của Nesterov accelerated gradient

đại lượng này:

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

Trong đó γ thường chọn là một giá trị khoảng 0.9, v_{t-1} chính là vận tốc tại thời điểm trước đó, $\nabla_\theta J(\theta)$ 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. Qủa 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 $\gamma < 1$.

1.3 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:

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

$$\theta = \theta - v_t$$

1.4 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:

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

Trong đó, $G_t \in \mathbb{R}^{dxd}$ 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 ε 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.

1.5 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 θ 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:

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

$$\Rightarrow \Delta\theta = \frac{\partial f/\partial\theta}{\partial^2 f/\partial\theta^2} \Rightarrow \frac{1}{\frac{\partial^2 f}{\partial\theta^2}} = \frac{\Delta\theta}{\frac{\partial f}{\partial\theta}}$$

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.

1.6 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:

$$E[g^2]_t = 0.9[g^2]_{t-1} + 0.1_t^2$$

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

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 $g_t \odot g_t$, 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**
    $t \leftarrow t + 1$;
    $g_t \leftarrow \nabla_\theta(\theta_{t-1})$;
    $m_t \leftarrow \beta_1.m_{t-1} + (1 - \beta_1).g_t$ (Update biased first moment estimate);
    $v_t \leftarrow \beta_2.v_{t-1} + (1 - \beta_2).g_t^2$ (Update biased second raw moment estimate) ;
    $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate) ;
    $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate) ;
    $\theta_t \leftarrow \theta_{t-1} - \alpha.\hat{m}_t/(\sqrt{\hat{v}_t} + \varepsilon)$ (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:

$$\begin{cases} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon} \end{cases} \tag{2.1}$$

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

$$\eta_t = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

$$\theta_t = \theta_{t-1} - \eta_t \frac{m_t}{\sqrt{v_t}+\varepsilon}$$

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 $v_k = v_k(0) = E(X^k)$ và gọi nó là moment gốc cấp k. Rõ ràng kỳ vọng chính là moment gốc cấp 1 $EX = v_1$. 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 $VX = \mu_2$
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

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 m_t và v_t 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:

$$\mathbb{E}[m_t] = \mathbb{E}[g_t]$$

$$\mathbb{E}[v_t] = \mathbb{E}[g_t^2]$$

Ta khai triển v_t như sau (tương tự với m_t):

$$\begin{aligned}
&v_0 = 0 \\
&v_1 = \beta_2 v_0 + (1 - \beta_2) g_1^2 = (1 - \beta_2) g_1^2 \\
&v_2 = \beta_2 v_1 + (1 - \beta_2) g_2^2 = \beta_2 (1 - \beta_2) g_1^2 + (1 - \beta_2) g_2^2 \\
&... \\
&v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} . g_i^2
\end{aligned}$$

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 v_t, để xem nó liên quan thế nào đến giá trị thực moment cấp 2 là $\mathbb{E}[g_t^2]$. Ta có:

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

Ở khai triển trên, chúng ta đã sử dụng xấp xỉ g_i thành g_t 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 m_t. 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 xải của Adam

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

a. Độ lớn tối đa của bước xải xấp xỉ η

b. 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 Đánh giá sự hội tụ

2.3.1 Sai lầm trong chứng minh hội tụ của Kingma & Ba

Trong bài báo gốc của mình giới thiệu về thuật toán Adam, Kingma và Lei Ba đã có những sai lầm trong phân tích sự hội tụ của thuật toán Adam. Sau đây, chúng ta sẽ cùng phân tích lỗi sai của họ.
Gọi $f_1(\theta), f_2(\theta), ..., f_T(\theta)$ là một dãy các hàm chi phí và lồi tại vòng lặp tương ứng $1, 2, ..., T$.
Đặt $g_t := \nabla f_t(\theta_t)$ và gọi $g_{t,i}$ là phần tử thứ i của vector gradient g_t.
Đặt $g_{1:t,i} = [g_{1,i}, g_{2,i}, ..., g_{t,i}] \in \mathbb{R}^t$ và $\gamma := \frac{\beta_1^2}{\sqrt{\beta_2}}$.
Gọi θ^* là tham số tối ưu của bài toán, θ_t là tham số dự đoán được tại vòng lặp thứ t. Do đó, tổng sai lệch của mô hình qua T vòng lặp là:

$$R(T) = \sum_{t=1}^{T} [f_t(\theta_t) - f_t(\theta^*)]$$

Rõ ràng, chúng ta cần chứng minh cận trên của $R(T)$ sẽ nhỏ hơn hoặc bằng $O(T)$ để từ đó có thể đánh giá được $\frac{R(T)}{T} \to C$ khi $T \to \infty$. Trong bài báo của mình, các tác giả đã chứng minh và chỉ ra định lý sau:

Định lý 1. *Giả sử rằng các hàm f_t là có gradient bị chặn, $||\nabla f_t(\theta)||_2 \leq G$, $||\nabla f_t(\theta)||_\infty \leq G_\infty$ với mọi tham số $\theta \in \mathbb{R}^d$, và khoảng cách giữa 2 tham số θ_t bất kỳ được sinh ra bởi Adam là bị chặn, $||\theta_n - \theta_m||_2 \leq D$, $||\theta_m - \theta_n||_\infty \leq D_\infty$ với mọi m, n bất kỳ thuộc $\{1, ..., T\}$, và $\beta_1, \beta_2 \in [0, 1)$ thoả mãn $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$. Đặt $\alpha_t = \frac{\alpha}{\sqrt{t}}$ và $\beta_{1,t} = \beta_1 \lambda^{t-1}, \lambda \in (0, 1)$. Thì ta có:*

$$R(T) \leq \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\hat{v}_{T,i}} + \frac{\alpha(1+\beta_1)G_\infty}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{d} ||g_{1:T,i}||_2 + \sum_{i=1}^{d} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\alpha(1-\beta_1)(1-\lambda)^2}$$

Từ định lý này, các tác giả kết luận được rằng $\frac{R(T)}{T} = O(\frac{1}{\sqrt{T}})$. Từ đó, khi $T \to \infty$ thì $\frac{R(T)}{T} \to 0$ và hội tụ. Tuy nhiên, định lý trên là không chính xác vì các tác giả đã chứng minh sai các bổ đề trong bài báo. Các tác giả đã sử dụng các bổ đề sau:

Bổ đề 1. *Nếu một hàm $f : \mathbb{R}^d \to \mathbb{R}$ là lồi, thì với mọi $x, y \in \mathbb{R}^d$, với mọi $\lambda \in [0, 1]$, thì:*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$

Bổ đề 2. *Giả sử $||g_t||_2 \leq G, ||g_t||_\infty \leq G_\infty$, thì:*

$$\sum_{t=1}^{T} \sqrt{\frac{g_{t,i}^2}{t}} \leq 2G_\infty ||g_{1:T,i}||_2$$

Bổ đề 3. *Giả sử $\gamma < 1$ và $||g_t||_2 \leq G$, $||g_t||_\infty \leq G_\infty$, thì:*

$$\sum_{t=1}^{T} \frac{\hat{m}_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}} \leq \frac{2}{1-\gamma} \frac{1}{\sqrt{1-\beta_2}} ||g_{1:T,i}||_2$$

Bổ đề 1 là một tính chất của hàm lồi, phổ biến và đã được chứng minh nhiều trong các tài liệu khác nên chúng ta thừa nhận kết quả này. Sau đây là chứng minh chi tiết của Kingma và Lei Ba và lỗi sai trong chứng minh các bổ đề trong báo cáo của họ:

Chứng minh bổ đề 2. Chúng ta sẽ chứng minh bằng phương pháp quy nạp.
Với $T = 1$, ta có $\sqrt{g_{1,i}^2} = |g_{1,i}| \leq G_\infty$.
Tuy nhiên, trong báo cáo của mình, Kingma & Ba đã đánh giá như sau:

$$\sqrt{g_{1,t}^2} \leq 2G_\infty ||g_{1,i}||_2$$

Rõ ràng, điều này chỉ đúng khi $G_\infty \geq 0.5$ và điều kiện này không được các tác giả đưa ra. Hơn nữa, bước chứng minh quy nạp cũng có một sai lầm khác trong bổ đề này.
Giả sử bổ đề đúng với $T = K - 1$, ta có:

$$\sum_{t=1}^{K-1} \sqrt{\frac{g_{t,i}^2}{t}} \leq 2G_\infty ||g_{1:K-1,i}||_2$$

Cần chứng minh, bổ đề đúng với $T = K$:

$$\begin{aligned}\sum_{t=1}^{K}\sqrt{\frac{g_{t,i}^2}{t}} &= \sum_{t=1}^{K-1}\sqrt{\frac{g_{t,i}^2}{t}} + \sqrt{\frac{g_{K,i}^2}{K}} \\ &\leq 2G_\infty||g_{1:K-1,i}||_2 + \sqrt{\frac{g_{K,i}^2}{K}} \\ &= 2G_\infty\sqrt{\sum_{t=1}^{K-1} g_{t,i}^2 + g_{K,i}^2 - g_{K,i}^2} + +\sqrt{\frac{g_{K,i}^2}{K}} \\ &= 2G_\infty\sqrt{||g_{1:K,i}||_2^2 - g_{K,i}^2} + +\sqrt{\frac{g_{K,i}^2}{K}}\end{aligned}$$

Sử dụng bất đẳng thức:

$$||g_{1:K,i}||_2^2 - g_{K,i}^2 \leq ||g_{1:K,i}||_2^2 - g_{K,i}^2 + \frac{g_{K,i}^4}{4||g_{1:K,i}||_2^2} = \left(||g_{1:K,i}||_2 - \frac{g_{K,i}^2}{2||g_{1:K,i}||_2}\right)^2$$

Từ đó:

$$\begin{aligned}\sum_{t=1}^{K}\sqrt{\frac{g_{t,i}^2}{t}} &\leq 2G_\infty\left(||g_{1:K,i}||_2 - \frac{g_{K,i}^2}{2||g_{1:K,i}||_2}\right) + \sqrt{\frac{g_{K,i}^2}{K}} \\ &\leq 2G_\infty\left(||g_{1:K,i}||_2 - \frac{g_{K,i}^2}{2\sqrt{KG_\infty^2}}\right) + \sqrt{\frac{g_{K,i}^2}{K}} \\ &= 2G_\infty||g_{1:K,i}||_2 - \frac{g_{K,i}^2}{\sqrt{K}} + \sqrt{\frac{g_{K,i}^2}{K}}\end{aligned}$$

Trong chứng minh của Kingma & Ba họ đánh giá như sau:

$$2G_\infty||g_{1:K,i}||_2 - \frac{g_{K,i}^2}{\sqrt{K}} + \sqrt{\frac{g_{K,i}^2}{K}} \leq 2G_\infty||g_{1:T,i}||_2$$

Nếu đẳng thức trên đúng, thì quy nạp sẽ kết thúc chứng minh. Tuy nhiên, không đảm bảo rằng đẳng thức trên sẽ đúng, đẳng thức trên tương đương với:

$$g_{K,i}^2 \leq |g_{K,i}|$$

chỉ đúng với $|g_{K,i}| \leq 1$. Do đó, bổ để 1, được dùng để chứng minh cho bổ đề 2 là chưa chặt chẽ. □

Chứng minh bổ đề 3. Từ giả thiết, ta có $\frac{\sqrt{1-\beta_2^t}}{(1-\beta_1^t)^2} \leq \frac{1}{(1-\beta_1)^2}$. Khai triển vế trái, ta có:

$$\begin{aligned}
\sum_{t=1}^{T} \frac{\hat{m}_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}} &= \sum_{t=1}^{T-1} \frac{\hat{m}_{t,i}}{\sqrt{t\hat{v}_{t,i}}} + \frac{\sqrt{1-\beta_2^T}}{(1-\beta_1^T)^2} \frac{(\sum_{k=1}^{T}(1-\beta_1)\beta_1^{T-k} g_{k,i})^2}{\sqrt{T\sum_{j=1}^{T}(1-\beta_2)\beta_2^{T-j} g_{j,i}^2}} \\
&\leq \sum_{t=1}^{T-1} \frac{\hat{m}_{t,i}}{\sqrt{t\hat{v}_{t,i}}} + \frac{\sqrt{1-\beta_2^T}}{(1-\beta_1^T)^2} \sum_{k=1}^{T} \frac{T((1-\beta_1)\beta_1^{T-k} g_{k,i})^2}{\sqrt{T\sum_{j=1}^{T}(1-\beta_2)\beta_2^{T-j} g_{j,i}^2}} \\
&\leq \sum_{t=1}^{T-1} \frac{\hat{m}_{t,i}}{\sqrt{t\hat{v}_{t,i}}} + \frac{\sqrt{1-\beta_2^T}}{(1-\beta_1^T)^2} \sum_{k=1}^{T} \frac{T((1-\beta_1)\beta_1^{T-k} g_{k,i})^2}{\sqrt{T(1-\beta_2)\beta_2^{T-k} g_{k,i}^2}} \\
&\leq \sum_{t=1}^{T-1} \frac{\hat{m}_{t,i}}{\sqrt{t\hat{v}_{t,i}}} + \frac{\sqrt{1-\beta_2^T}}{(1-\beta_1^T)^2} \frac{(1-\beta_1)^2}{\sqrt{T(1-\beta_2)^2}} \sum_{k=1}^{T} T\left(\frac{\beta_1^2}{\sqrt{\beta_2}}\right)^{T-k} ||g_{k,i}||_2 \\
&\leq \sum_{t=1}^{T-1} \frac{\hat{m}_{t,i}}{\sqrt{t\hat{v}_{t,i}}} + \frac{T}{\sqrt{T(1-\beta_2)}} \sum_{k=1}^{T} \gamma^{T-k} ||g_{k,i}||_2
\end{aligned}$$

Tới đây, trong báo cáo của mình, Kingma & Ba đã chứng minh như sau:

$$\begin{aligned}
\sum_{t=1}^{T} \frac{\hat{m}_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}} &\leq \sum_{t=1}^{T} \frac{||g_{t,i}||_2}{\sqrt{t(1-\beta_2)}} \sum_{j=0}^{T-t} t\gamma^j \\
&\leq \sum_{t=1}^{T} \frac{||g_{t,i}||_2}{\sqrt{t(1-\beta_2)}} \sum_{j=0}^{T} t\gamma^j
\end{aligned}$$

Điều này là không dễ dàng suy ra được. Rõ ràng, từ trên ta chỉ có thể đánh giá như sau:

$$\sum_{t=1}^{T} \frac{\hat{m}_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}} \leq \sum_{t=1}^{T} \frac{t}{\sqrt{t(1-\beta_2)}} \sum_{k=1}^{t} \gamma^{t-k} ||g_{k,i}||_2$$

Có lẽ các bước đánh giá sai bổ đề đã gây nhầm lẫn trong chứng minh hội tụ Adam của Kingma & Ba. □

2.3.2 Chứng minh thuật toán Adam có thể không hội tụ

Trong báo cáo của mình tại ICLR 2018, Sashank J. Reddi, Satyen Kate và Sanjiv Kumar đã chứng minh thuật toán Adam không hội tụ và đề xuất ra các mở rộng, cải tiến.

Các ký hiệu:

- $\mathbb{S}_d^+$: tập tất cả các ma trận xác định dương $d \times d$.
- $\mathbb{F}$ là tập không gian Euclid d chiều.
- Toán tử chiếu $\prod_{\mathbb{F},\mathbb{A}}(y)$ đã định nghĩa như $argmin_{x\in\mathbb{F}}||\mathbb{A}^{1/2}(x-y)||$ với mọi $y \in \mathbb{R}^d$.
- $\mathbb{F}$ có đường kính bị chặn nếu $||x-y||_\infty \leq D_\infty$ với mọi $x, y \in \mathbb{F}$.

Ý tưởng thuật toán gradient descent là giải quyết bài toán tối ưu:

$$x^* = argmin_{x\in\mathbb{F}}\mathbb{E}\left[f(x,z)\right]$$

Trong đó: $f(x,z)$ là hàm mất mát tham số x trên mẫu z.
Trong thực tế, phân phối của z là không biết, nhưng chúng ta có một tập N mẫu training $\{z\}_{n=1}^N$. Từ đó, đưa về bài toán tối ưu Empirical Risk Minimization (ERM):

$$x^* = argmin_{x\in\mathbb{F}}\frac{1}{N}\sum_{n=1}^{N} f(x,z_n)$$

Khung linh hoạt để phân tích sự hội tụ của các phương pháp tối ưu là sử dụng các thiết lập online. Trong thiết lập online, tại mỗi bước t, thuật toán lấy ra một điểm $x_t \in \mathbb{F}$, trong đó $\mathbb{F} \in \mathbb{R}^d$ là tập các điểm khả thi. Một hàm mất mát f_t là đã biết, và thuật toán phát sinh mất mát là $f_t(x_t)$. Tổng sai số của thuật toán tại bước kết thúc T được cho bởi:

$$R_T = \sum_{i=1}^{T} f_t(x_t) - min_{x\in\mathbb{F}}\sum_{i=1}^{T} f_t(x)$$

Trong phần này, chúng ta giả sử rằng $\mathbb{F}$ có đường kính bị chặn và $||\nabla f_t(x)||_\infty$ là bị chặn với mọi $t \in [T]$ và $x \in \mathbb{F}$. Mục tiêu của chúng ta là đưa ra một thuật toán đảm bảo $R_T = O(T)$, khi đó chúng ta có thế kết luận là thuật toán hội tụ. Thuật toán đơn giản nhất cho các thiết lập này là thuật toán gradient descent online (Zinkevick 2003) có quy tắc update là:

$$x_{t+1} = \prod_{\mathbb{F}}(x_t - \alpha_t g_t)$$

Trong đó, $\prod_{\mathbb{F}}$ biểu diễn phép chiếu $y \in \mathbb{R}^d$ trên tập $\mathbb{F}$, i.e $\prod_{x\in\mathbb{F}}(y) = min_{x\in\mathbb{F}}||x-y||$, và $\alpha_t = \alpha/\sqrt{t}$. Bài toán tối ưu online này có quan hệ gần gũi với gradient descent nói trên.

Thiết lập chung cho các phương pháp adaptive (thích nghi)

Các tác giả cung cấp một khung chung cho các phương pháp thích nghi cung cấp cách nhìn sâu sắc về sự khác biệt giữa các phương pháp thích nghi khác nhau và rất hữu ích để hiểu các sai sót trong một vài phương pháp thích ứng phổ biến.

Algorithm 2: Generic Adaptive Method Setup

Input: $x_1 \in \mathbb{F}$, step size $\{\alpha_t\}_t^T$, sequence of functions $\{\phi_t \psi\}$;
for $t = 1$ ***to*** T **do**
  $g_t = \nabla f_t(x_t)$;
  $m_t = \phi_t(g_1, ..., g_t)$ and $V_t = \psi_t(g_1, ..., g_t)$;
  $\hat{x}_{t+1} = x_t - \alpha_t m_t / \sqrt{V_t}$;
  $x_{t+1} = \prod_{\mathbb{F}, \sqrt{V_t}}(\hat{x}_{t+1})$;
end

Trong đó, ϕ_t và ψ_t được gọi là các hàm "averaging" và chưa xác định:

$$\phi_t : \mathbb{F}^t \to \mathbb{R}^d$$

$$\psi_t : \mathbb{F}^t \to \mathbb{S}_d^+$$

Để dễ trình bày, coi α_t như step size, $\alpha_t . V^{-1/2}$ như learning rate và $V_t = diag(v_t)$. Dễ thấy, thuật toán gradient descent nằm trong khung adaptive vì nó sử dụng:

$$\phi_t(g_1, ..., g_t) = g_t$$

$$\psi_t(g_1, ..., g_t) = \mathbb{I}$$

Và $\alpha_t = \frac{\alpha}{\sqrt{t}}$ với mọi $t \in [T]$. Ý tưởng chính của phương pháp thích nghi là chọn các hàm averaging một cách thích hợp để thuật toán hội tụ tốt. Ví dụ phương pháp thích nghi đầu tiên là AdaGrad, mở đầu cho sự phát triển các phương pháp adaptive sau này:

$$\phi_t(g_1, ..., g_t) = g_t$$

$$\psi_t(g_1, ..., g_t) = \frac{diag(\sum_{i=1}^t g_i^2)}{t}$$

Phương pháp thích nghi (adaptive) dựa trên trung bình trượt theo cấp số nhân (Exponential Moving Averages)

Trung bình trượt theo cấp số nhân khá phổ biến trong cộng đồng deep learing. RMSProp, Adam, Nadam và adadelta là một số thuật toán phổ biến sinh ra từ phạm trù này. Chìa khoá cho sự khác biệt của các phương pháp này là sử dụng trung bình trượt theo cấp số nhân như hàm ψ_t để thay thế cho hàm trung bình đơn giản sử dụng trong Adagrad.

Adam (để đơn giản, chúng ta bỏ công thức bias-correction trong phiên bản gốc của Adam) là một biến thể phổ biến, sử dụng các hàm averaging sau:

$$\phi_t(g_1, ..., g_t) = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-1} g_i$$

$$\psi_t(g_1, ..., g_t) = (1 - \beta_2) diag(\sum_{i=1}^{t-1} g_i^2)$$

Nói một cách khác, có thể viết đệ quy như sau:

$$m_{t,i} = \beta_1 m_{t-1,i} + (1 - \beta_1) g_{t,i}$$

$$v_{t,i} = \beta_2 v_{t-1,i} + (1 - \beta_2) g_{t,i}^2$$

$$m_{0,i} = 0, v_{0,i} = 0$$

với mọi $i \in [d]$ và $t \in [T]$. Các giá trị $\beta_1 = 0.9$ và $\beta_2 = 0.999$ được khuyến nghị. Khi $\mathbb{F} = \mathbb{R}^d$, toán tử chiếu trong thuật toán adaptive chung là toán tử đồng nhất và ta được thuật toán Adam tương đương với thuật toán trong bài báo gốc. Để phân tích lý thuyết, người ta yêu cầu $\alpha_t = \frac{\alpha}{\sqrt{t}}$ mặc dù lựa chọn step size cố định có vẻ tốt hơn trong thực tế.

Sự không hội tụ của thuật toán Adam

Các tác giả thấy rằng vấn đề chính dẫn đến sự không hội tụ của Adam nằm ở đại lượng sau:

$$\Gamma_{t+1} = \left(\frac{\sqrt{V_{t+1}}}{\alpha_{t+1}} - \frac{\sqrt{V_t}}{\alpha_t} \right)$$

Đại lượng này đo lường sự thay đổi của nghịch đảo learning rate cho các phương pháp thích nghi theo thời gian. Dễ thấy rằng với SGD và AdaGrad thì $\Gamma_t \geq 0$ với mọi $t \in [T]$. Tuy nhiên, điều này là chưa hẳn với các phương pháp sử dụng trung bình trượt theo cấp số nhân như Adam và RMSProp. Γ_t có thể bất ổn định với $t \in [T]$. Vi phạm này có thể dẫn đến sự không hội tụ của Adam và RMSProp.

Định lý 2. *Tồn tại một bài toán tối ưu lồi online mà thuật toán Adam là không hội tụ. tức là* $R(T)/T \nrightarrow 0$ *khi* $T \to \infty$.

Chứng minh. Chúng ta xét f_t là các hàm tuyến tính và $\mathbb{F} = [-1, 1]$. Xét dãy hàm:

$$f_t(x) = \begin{cases} Cx, & \text{Nếu t mod 3 = 1} \\ -x, & \text{Các trường hợp khác} \end{cases}$$

Với $C \geq 2$. Xét thuật toán Adam với các tham số sau:

$$\beta_1 = 0, \beta_2 = \frac{1}{1 + C^2}, \alpha_t = \frac{\alpha}{\sqrt{t}}, \alpha < \sqrt{1 - \beta_2}$$

Dễ thấy bài toán trên cùng với các tham số đã chọn thoả mãn điều kiện hội tụ trong bài báo gốc Adam. Dễ dàng thấy rằng $x = -1$ là điểm cực tiểu $R(T)$.
Chúng ta khởi tạo $x_1 = 1$ và sẽ chứng minh dãy $\{x_t\}_{t=1}^{\infty}$ sinh ra từ quy tắc cập nhật Adam luôn có $x_t > 0$ với mọi $t \in \mathbb{N}$, $x_{3t+1} = 1$ với mọi $t \in \mathbb{N} \cup \{0\}$. Chúng ta sẽ sử dụng phương pháp quy nạp.
Với $x_1 = 1$ thoả mãn cả 2 điều kiện.
Giả sử với $t \in \mathbb{N} \cup \{0\}$, chúng ta có $x_i > 0$ với mọi $i \in [3t+1]$ và $x_{3t+1} = 1$. Cần chứng minh $x_{3t+2} > 0$, $x_{3t+3} > 0$ và $x_{3t+4} = 1$.
Thật vậy, ta có:

$$\nabla f_i(x) = \begin{cases} C, & \text{Với i mod } 3 = 1 \\ -1, & \text{Các trường hợp khác} \end{cases}$$

Theo quy tắc cập nhật Adam đã xét, ta có:

$$\begin{aligned} \hat{x}_{3t+2} &= x_{3t+1} - \frac{\alpha C}{\sqrt{(3t+1)(\beta_2 v_{3t} + (1-\beta_2)C^2}} = 1 - \frac{\alpha C}{\sqrt{(3t+1)(\beta_2 v_{3t} + (1-\beta_2)C^2}} \\ &\geq 1 - \frac{\alpha C}{\sqrt{(3t+1)(1-\beta_2)C^2}} = 1 - \frac{\alpha}{\sqrt{(3t+1)(1-\beta_2)}} > 0 \end{aligned}$$

Bất đẳng thức cuối xảy ra do $\alpha < \sqrt{1-\beta_2}$. Từ đó, ta có $1 > x_{3t+2} = \hat{x}_{3t+2} > 0$. Tương tự, ta có:

$$\begin{aligned} \hat{x}_{3t+3} &= x_{3t+2} + \frac{\alpha}{\sqrt{(3t+2)(\beta_2 v_{3t+1} + (1-\beta_2))}} \\ \hat{x}_{3t+4} &= x_{3t+3} + \frac{\alpha}{\sqrt{(3t+3)(\beta_2 v_{3t+2} + (1-\beta_2))}} \end{aligned}$$

Do $x_{3t+2} > 0$ nên dễ ràng có được $x_{3t+3} > 0$. Để hoàn thành chứng minh, chúng ta cần chứng minh $x_{3t+4} = 1$. Ở đây, ta sẽ cố gắng chứng minh $\hat{x}_{3t+4} \geq 1$. Từ đó dễ dàng có được $x_{3t+4} = 1$ vì $x_{3t+4} = \prod_{\mathbb{F}}(\hat{x}_{3t+4})$ và $\mathbb{F} = [-1, 1]$ với $\prod_{\mathbb{F}}$ đơn giản là toán tử chiếu Euclid (Trong không gian 1 chiều thì $\prod_{\mathbb{F},\sqrt{V_t}} = \prod_{\mathbb{F}}$). Ta có:

$$\hat{x}_{3t+4} = min(\hat{x}_{3t+3}, 1) + \frac{\alpha}{\sqrt{(3t+3)(\beta_2 v_{3t+2} + (1-\beta_2))}}$$

Đẳng thức trên xảy ra do $\hat{x}_{3t+3}$ và toán tử chiếu trên miền $\mathbb{F} = [-1, 1]$.
Xét 2 trường hợp: 1. Giả sử $\hat{x}_{3t+3} \geq 1$, thì dễ dàng có được $\hat{x}_{3t+4} > 1$.
2. Giả sử $\hat{x}_{3t+3} < 1$, ta có:

$$\begin{aligned} \hat{x}_{3t+4} &= \hat{x}_{3t+3} + \frac{\alpha}{\sqrt{(3t+3)(\beta_2 v_{3t+2} + (1-\beta_2))}} \\ &= x_{3t+2} + \frac{\alpha}{\sqrt{(3t+2)(\beta_2 v_{3t+1} + (1-\beta_2)}} + \frac{\alpha}{\sqrt{(3t+3)(\beta_2 v_{3t+2} + (1-\beta_2))}} \\ &= 1 - \frac{\alpha C}{\sqrt{(3t+1)(\beta_2 v_{3t} + (1-\beta_2)C^2}} + \frac{\alpha}{\sqrt{(3t+2)(\beta_2 v_{3t+1} + (1-\beta_2))}} \\ &+ \frac{\alpha}{\sqrt{(3t+3)(\beta_2 v_{3t+2} + (1-\beta_2))}} \end{aligned}$$

Đẳng thức cuối thứ 3 xảy ra do $x_{3t+2} = \hat{x}_{3t+2}$. Từ đó, để chứng minh $\hat{x}_{3t+4} > 1$, chúng ta cần chứng minh:

$$T_1 := \frac{\alpha C}{\sqrt{(3t+1)(\beta_2 v_{3t} + (1-\beta_2)C^2}} \leq \frac{\alpha}{\sqrt{(3t+2)(\beta_2 v_{3t+1} + (1-\beta_2))}}$$

$$+ \frac{\alpha}{\sqrt{(3t+3)(\beta_2 v_{3t+2} + (1-\beta_2))}} := T_2$$

Chúng ta đã có:

$$T_1 \leq \frac{\alpha}{\sqrt{(3t+1)(1-\beta_2)}} \tag{2.2}$$

Xét T_2, ta có:

$$T_2 = \frac{\alpha}{\sqrt{(3t+2)(\beta_2 v_{3t+1} + (1-\beta_2))}} + \frac{\alpha}{\sqrt{(3t+3)(\beta_2 v_{3t+2} + (1-\beta_2))}} \tag{2.3}$$

$$\geq \frac{\alpha}{\sqrt{\beta_2 C^2 + (1-\beta_2))}} \left(\frac{1}{\sqrt{3t+2}} + \frac{1}{\sqrt{3t+3}} \right) \tag{2.4}$$

$$\geq \frac{\alpha}{\sqrt{\beta_2 C^2 + (1-\beta_2))}} \left(\frac{1}{\sqrt{2(3t+1)}} + \frac{1}{\sqrt{2(3t+1)}} \right) \tag{2.5}$$

$$= \frac{\sqrt{2}\alpha}{\sqrt{(3t+1)(\beta_2 C^2 + (1-\beta_2))}} = \frac{\alpha}{\sqrt{(3t+1)(1-\beta_2)}} \geq T_1 \tag{2.6}$$

Đẳng thức đầu tiên xảy ra do $v_t \leq C^2$ với mọi $t \in \mathbb{N}$. Đẳng thức cuối xảy ra do:

$$\sqrt{\frac{\beta_2 C^2 + (1-\beta_2)}{2}} = \sqrt{1-\beta_2}$$

vì chúng ta đã chọn $\beta_2 = 1/(1+C^2)$. Từ đó, chúng ta có $T_2 \leq T_1$ và $\hat{x}_{3t+4} \leq 1$.
Vậy trong cả 2 trường hợp thì $x_{3t+4} = 1$. Nên giả thuyết quy nạp đúng. Do vậy, ta có: $f_{3t+1}(x_{3t+1}) + f_{3t+2}(x_{3t+2}) + f_{3t+3}(x_{3t+3}) - f_{3t+1}(-1) - f_{3t+2}(-1) - f_{3t+3}(-1) \geq 2C - 4$. Do đó, cứ mỗi 3 bước, Adam bị mất mát 1 lượng $2C-4$, Hay $R(T) \geq (2C-4)T/3$. Vì $C \geq 2$, nên mất mát sẽ lớn dần và $R_T/T \nrightarrow 0$ khi $T \to \infty$. Từ đó được điều phải chứng minh. □

Chứng minh tương tự như trên, trong báo cáo của Reddi và các cộng sự cũng chứng minh được nếu có thêm ε ở công thức cập nhật Adam. Tức là:

$$\hat{x}_{t+1} = x_t - \alpha_t m_t / \sqrt{V_t + \varepsilon \mathbb{I}}$$

thì thuật toán vẫn có thể không hội tụ.

2.4 Các mở rộng của thuật toán Adam

2.4.1 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 3: *Adamax*, 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) ;
$u_0 \leftarrow 0$ (Initialize the exponentially weighted infinit norm) ;
$t \leftarrow 0$ (Initialize timestep) ;
while θ_t *not converged* **do**
  $t \leftarrow t + 1$;
  $g_t \leftarrow \nabla_\theta(\theta_{t-1})$;
  $m_t \leftarrow \beta_1.m_{t-1} + (1-\beta_1).g_t$ (Update biased first moment estimate);
  $u_t \leftarrow max(\beta_2.u_{t-1}, |g_t|)$ (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 $p \to \infty$ và định nghĩa $u_t = lim_{p\to\infty}(v_t)^{1/p}$, thì:

$$\begin{aligned}
u_t = lim_{p\to\infty}(v_t)^{1/p} &= \lim_{p\to\infty}\left((1-\beta_2^p)\sum_{i=1}^{t}\beta_2^{p(t-i)}.|g_i|^p\right)^{1/p} \\
&= \lim_{p\to\infty}(1-\beta_2^p)^{1/p}\left(\sum_{i=1}^{t}\beta_2^{p(t-i)}.|g_i|^p\right)^{1/p} \\
&= \lim_{p\to\infty}\left(\sum_{i=1}^{t}\beta_2^{p(t-i)}.|g_i|^p\right)^{1/p} \\
&= max(\beta_2^{t-1}|g_1|, \beta_2^{t-2}|g_2|, ..., \beta_2|g_{t-1}|, |g_t|)
\end{aligned}$$

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

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

2.4.2 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 m_t trong Adam.
Đầu tiên, ta xem lại công thức cập nhật momentum:

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

Trong đó, $\nabla\theta_t J(\theta_t)$ 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:

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

Dozat đề xuất cải biến NAG như sau: Thay vì áp dụng momentum hai lần - một lần để cập nhật gradient g_t và lần thứ hai để cập nhật θ_{t+1}. Giờ chúng ta áp dụng trực tiếp momentum xấp xỉ điểm tiếp theo để cập nhật các tham số hiện tại:

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

Có thể thấy trong công thức trên, thay vì sử dụng vector momentum m_{t-1} như trong phương pháp momentum, giờ ta sử dụng vector momentum m_t 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:

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

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 $\hat{m}_{t-1}$ bằng chỉ số bias-corrected của vector momentum hiện tại $\hat{m}_t$. Từ đó, ta có công thức cập nhật của Nadam:

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

2.4.3 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 v_t 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:

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

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

3.1 Thực nghiệm Adam và các thuật toán cơ sở

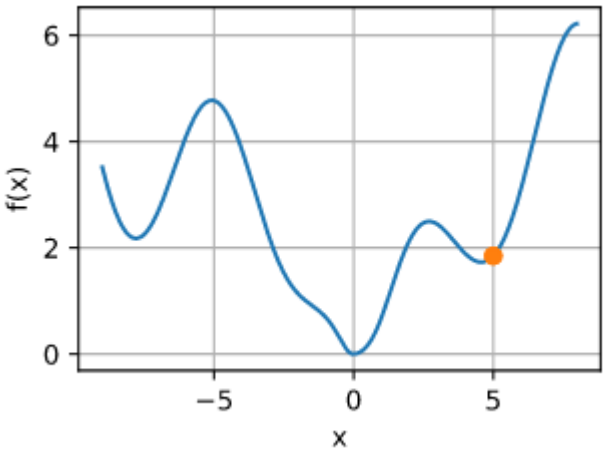
Các thuật toán Gradient Descent có rất nhiều vấn đề trong quá trình hội tụ , đặc biệt với những hàm số khó hội tụ tốt như hàm số có nhiều cực tiểu địa phương , nhiều điểm yên ngựa .
Để so sánh Adam đã giải quyết những vấn đề trên so với những giải thuật cơ sở ta tiến hành thử nghiệm trên một số hàm số đặc biệt :

- hàm số $f(x) = log(1 + (\mid x \mid)^{2+sinx}))$ (hàm số 1)
- hàm số $f(x) = (2 + \frac{sin50x}{50})(arctanx)^2$ (hàm số 2)
- Neural network với bộ dữ liệu mnist

3.1.1 Hàm số 1

$$f(x) = log(1 + (\mid x \mid)^{2+sinx})$$

Đồ thị :



Hàm số trong khoảng -9 đến 9 có 3 điểm cực tiểu .

Với khởi tạo ban đầu $x_0 = 2, \alpha = 0.01, \beta_1 = 0.9, \beta_2 = 0.99$, kết quả của các thuật toán đạt được :

```
epoch  868 x: [-1.06588016e-08]
Gradient descent [-1.06588016e-08]
epoch 265 x [-2.17762022e-06]
Momentum [-2.17762022e-06]
epoch 267 x [-8.78597066e-07]
Nag [-8.78597066e-07]
epoch 9999 x [0.33193342]
Adagrad [0.33193342]
epoch 243 x [-4.99176985e-07]
RMSprop [-4.99176985e-07]
Adam: epoch 569 x [-7.69715167e-07]
Adam [-7.69715167e-07]
```

Có thể thấy với khởi tạo ban đầu lý tưởng các thuật toán đều có thể hội tụ tốt , với thời gian hội tụ ngắn .
Với khởi tạo khó hơn $x_0 = 8, \alpha = 0.01, \beta_1 = 0.9, \beta_2 = 0.99$, kết quả của các thuật toán :

```
epoch  1433 x: [4.56611984]
Gradient descent [4.56611984]
epoch 296 x [4.56612179]
Momentum [4.56612179]
epoch 302 x [4.56611949]
Nag [4.56611949]
epoch 9999 x [5.64208075]
Adagrad [5.64208075]
epoch 382 x [4.56611898]
RMSprop [4.56611898]
Adam: epoch 1110 x [4.56611936]
Adam [4.56611936]
```

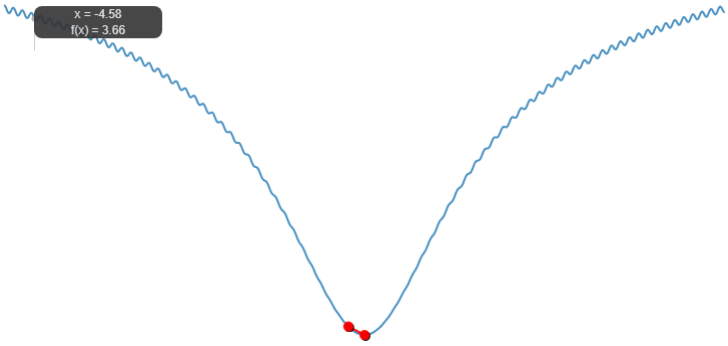
Có thể thấy các thuật toán đều hội tụ đến điểm local minimum và không thể tới được điểm global minimum . Ta tăng α lên 2 , kết quả :

```
epoch  9999 x: [5.0106917]
Gradient descent [5.0106917]
epoch 9999 x [31166.58848496]
Momentum [31166.58848496]
epoch 9999 x [12.18847365]
Nag [12.18847365]
epoch 12 x [4.56611898]
Adagrad [4.56611898]
epoch 9999 x [0.90725303]
RMSprop [0.90725303]
Adam: epoch 303 x [-8.4867482e-07]
Adam [-8.4867482e-07]
```

Duy nhất Adam có thể tiến tới điểm global minimum nhờ hiệu ứng Heavy Ball with Friction

3.1.2 Hàm số 2

$$f(x) = (2 + \frac{sin50x}{50})(arctanx)^2$$



Đồ thị :

Với khởi tạo ban đầu : $x_0 = 4.03, \alpha = 0.06, \beta_1 = 0.9, \beta_2 = 0.99$ kết quả đạt được :

```
epoch  9999 x: [2.08558813]
Gradient descent [2.08558813]
epoch 317 x [-6.59583539e-07]
Momentum [-6.59583539e-07]
epoch 179 x [-1.52504801e-06]
Nag [-1.52504801e-06]
epoch 50 x [3.98626395]
Adagrad [3.98626395]
epoch 158 x [-4.97920382e-07]
RMSprop [-4.97920382e-07]
Adam: epoch 237 x [3.98626386]
Adam [3.98626386]
```

Trường hợp này các thuật toán khác có thể hội tụ tốt nhưng adam lại không qua được local minimum do điểm khởi tạo ban đầu quá gần với local minimum mà tốc độ học ban đầu lại quá nhỏ dẫn tới "ma sát" quá lớn khiến cho điểm k vượt qua được local minimum

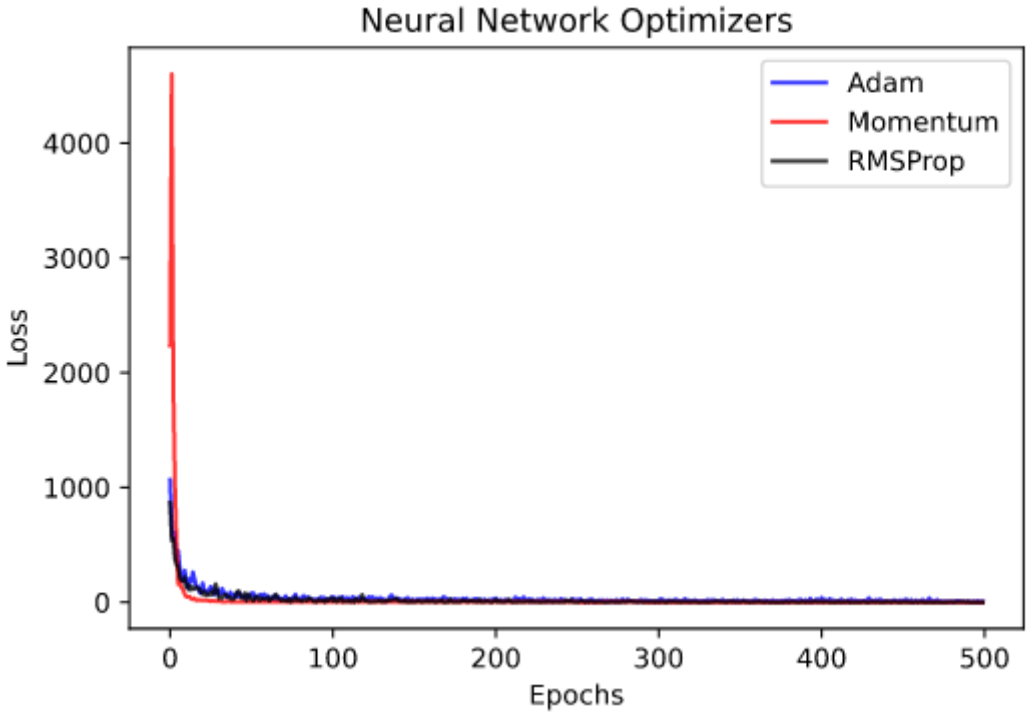
Tăng learning rate lên 2 và giữ nguyên những hyper-parameter khác thì kết quả :

```
epoch  9999 x: [-456.60608974]
Gradient descent [-456.60608974]
epoch 9999 x [930.83453448]
Momentum [930.83453448]
epoch 9999 x [-119.53706345]
Nag [-119.53706345]
epoch 148 x [-5.04527632e-07]
Adagrad [-5.04527632e-07]
epoch 9999 x [329.12282844]
RMSprop [329.12282844]
Adam: epoch 346 x [-5.26884698e-07]
Adam [-5.26884698e-07]
```

Vì learning rate quá lớn nên trong 10000 epoch momentum chưa kịp hội tụ còn adam đã hội tụ tốt

3.1.3 Neural network

Thí nghiệm so sánh các thuật toán tối ưu sử dụng bộ dữ liệu chữ viết tay MNIST .
Bài toán phân loại các vector ảnh 784 chiều vào 10 lớp
Sử dụng 1 neural network với 2 hidden layer 256 unit .
Hyper-parameter : $\beta_1 = 0.9, \beta_2 = 0.99, \alpha = 0.01$, batch_size=128 , epoch = 500



3.2 Thực nghiệm Adam và AMSGrad

Thuật toán ADAM sử dụng các bình phương trung bình trượt (EMA) gradient trong quá khứ để cập nhật, điều này khiến các cập nhật sẽ bị ảnh hưởng bởi những gradient gần nhất, trong khi tầm quan trọng của những gradient cũ hơn sẽ nhanh chóng mất đi. Đây là nguyên nhân dẫn đến sự hội tụ kém của ADAM trong một số trường hợp.
Giải quyết vấn đề này là mục đích chính của bài báo "On The Convergence Of ADAM And Beyond". Tác giả đề xuất một giải thuật sử dụng một "bộ nhớ lâu dài" dành cho các gradient, trong đó các cập nhật sẽ sử dụng những gradient lớn nhất, có tầm quan trọng nhất trong các gradient quá khứ. Giải thuật có tên AMSGRAD.
Bài báo đưa ra ba thí nghiệm để so sánh giải thuật ADAM và AMSGRAD:

1 Synthetic

2 Logistic Regression

3 Neural Network

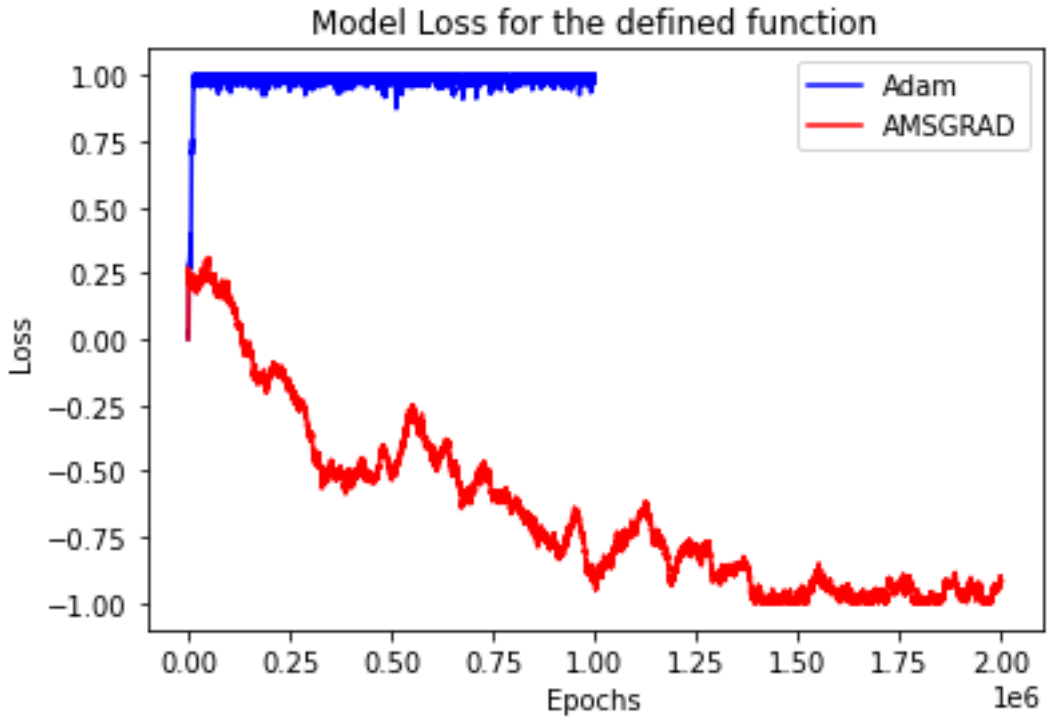
4 VGG

Thí nghiệm Synthetic có mục đích minh họa khả năng hội tụ của ADAM và AMSGRAD trong một cài đặt lồi đơn giản. Hai thí nghiệm còn lại để chỉ ra sự vượt trội của AMSGRAD so với ADAM khi áp dụng với dữ liệu trong thực tế. Dưới đây là cài đặt và kết quả thí nghiệm của nhóm:

3.2.1 Synthetic

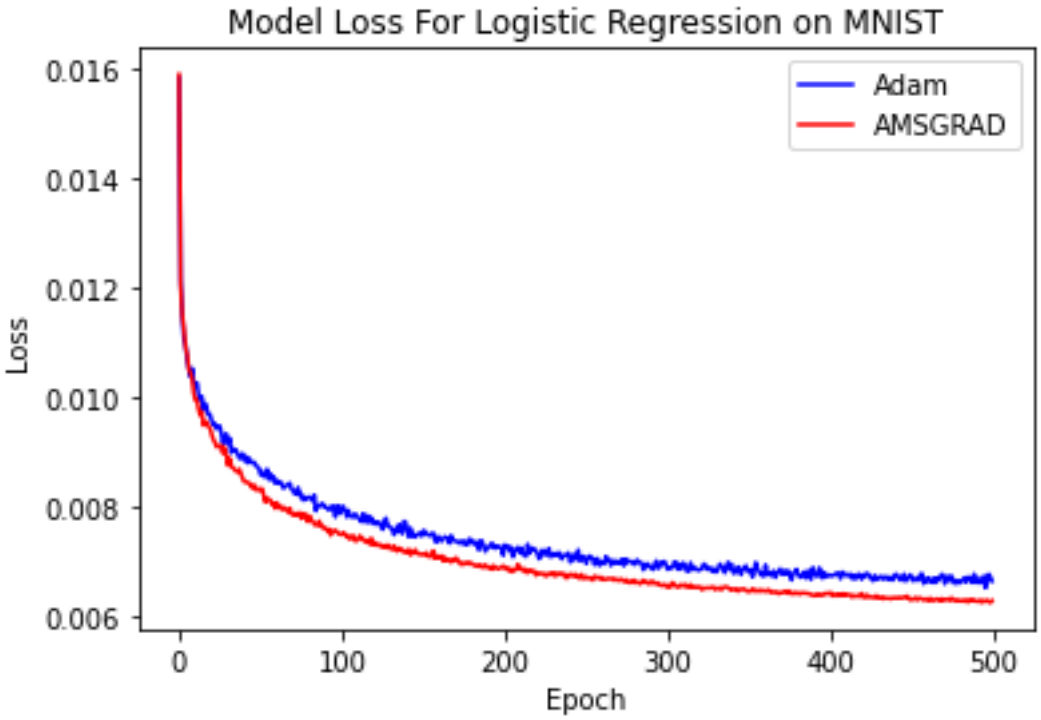
$$f_t(x) = \begin{cases} 1010x & \text{for } t \bmod 101 = 1 \\ -10x & \text{otherwise} \end{cases}$$

Với $\mathcal{F} = [-1; 1]$, có thể thấy giải pháp tối ưu là x = -1 vì tại điểm này sẽ cho ra regret nhỏ nhất. Do đó, x = -1 là điểm hội tụ tối ưu nhất.
Tuy nhiên, như đã thấy trong bài báo cũng như kết quả thí nghiệm dưới, ADAM lại hội tụ về x = 1. Mặt khác, AMSGRAD hội tụ về x = -1. Nguyên nhân là vì những gradient hữu ích cho hội tụ tối ưu chỉ xuất hiện với xác suất 0.01, cho nên ADAM không thể sử dụng các thông tin này một cách hiệu quả.
Hyper-parameters: $\beta_1 = 0.9, \beta_2 = 0.99, \alpha = 0.001, \text{batch size} = 1, \text{epochs} = 2M$



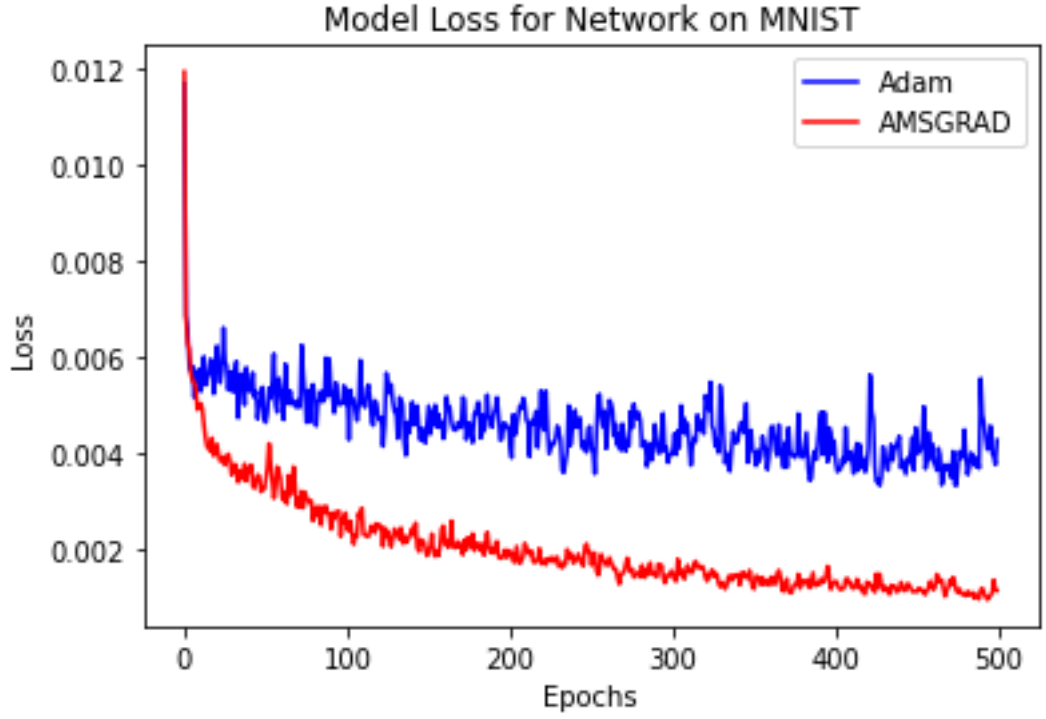
3.2.2 Logistic Regression

Thí nghiệm này sẽ áp dụng hai thuật toán tối ưu ADAM và AMSGRAD vào bài toán cổ điển nhận dạng chữ viết tay dựa trên bộ dữ liệu MNIST. Đồ thị so sánh hàm mất mát của hai thuật toán được đưa ra ở dưới. Bài toán sẽ phân loại các vector hình ảnh với 784 chiều vào 10 lớp label khác nhau.
Có thể thấy sự hội tụ của thuật toán AMSGRAD là tốt hơn so với ADAM.
Hyper-parameters: $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\alpha = 0.01$, batch size $= 128$, epochs $= 500$.



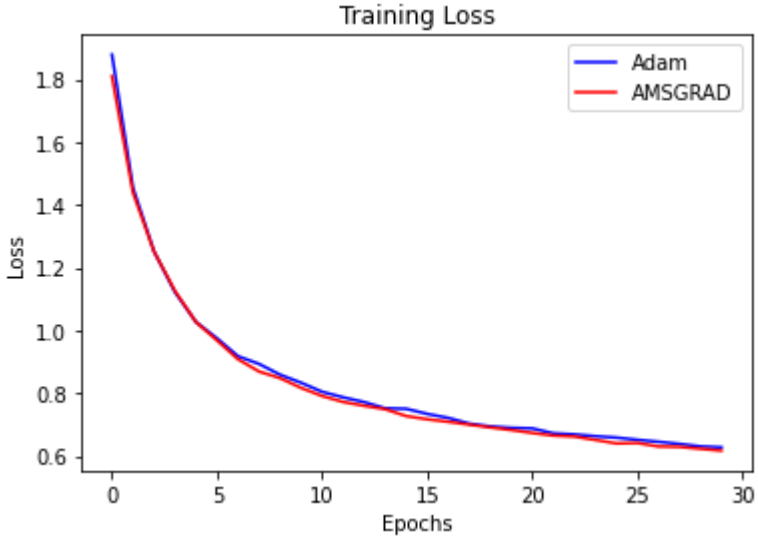
3.2.3 Neural Network

Tương tự như thí nghiệm 2, nhưng ta thêm một hidden layer với 100 neuron. Có thể thấy AMSGRAD một lần nữa lại cho ra kết quả tốt hơn ADAM.
Hyper-parameters: $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\alpha = 0.01$, batch size $= 128$, epochs $= 500$.

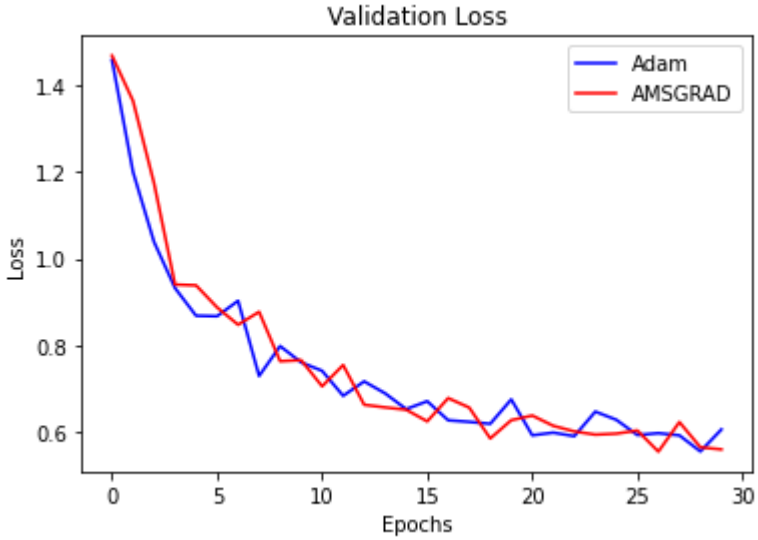


3.2.4 VGG

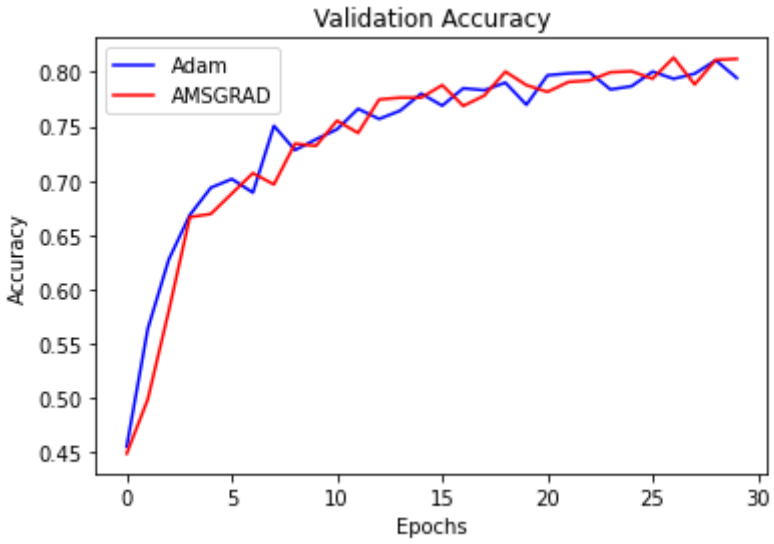
Trong một thí nghiệm khác sử dụng mạng VGG cho bộ dữ liệu cifar10 với Hyper parameters : $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\alpha = 0.001$, batch size $= 128$, epochs $= 30$



Có thể thấy AMSGrad hội tụ nhanh hơn ADAM trong phần lớn quá trình huấn luyện, tuy nhiên khi kết thúc huấn luyện cả hai thuật toán đều hội tụ về cùng một mất mát tương tự nhau. Thêm vào đó, tại một số lần lặp gần cuối, ADAM hội tụ nhanh hơn cả AMSGRAD. Những điều này không mang nhiều ý nghĩa vì đây chỉ là mất mát khi huấn luyện. Tuy nhiên, chúng lại khác với những kết luận được đưa ra trong bài báo 'On The Convergence Of ADAM And Beyond' khi huấn luyện với mô hình cifarnet, trong đó các tác giả kết luận răng train loss của AMSGRAD thấp hơn nhiều so với ADAM.



Validation loss lại cho ra kết quả khác xa Train loss. Có thể thấy, trong những lần lặp đầu, ADAM cho validation loss thấp hơn so với AMSGRAD. Ở những lần lặp sau đó, hai thuật toán liên tục đổi vị trí cho nhau khi so sánh giá trị validation loss.



Đối với validation accuracy, thuật toán ADAM cho độ chính xác cao hơn so với AMSGRAD trong phần lớn quá trình nhưng không ổn định. Và cũng như validation loss, độ chính xác của hai thuật toán này liên tục đổi ngôi cho nhau trong những lần lặp cuối.

Kết luận: Trong bài báo 'On The Convergence Of ADAM And Beyond', các tác giả kết luận rằng AMSGRAD cho hiệu năng tốt hơn nhiều so với ADAM khi so sánh training loss và accuracy. Tuy nhiên, những kết quả của thí nghiệm ở trên lại không cho thấy điều tương tự. ADAM mang lại hiệu năng tương đương và tại một số điểm còn tốt hơn AMSGRAD.

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.

Dù ADAM rất thông dụng trong thực tế, về mặt lý thuyết, cũng như trong một số thực nghiệm mà nhóm đã thực hiện, thuật toán này không hội tụ về điểm lý tưởng hoặc có khả năng hội tụ kém trong một số trường hợp. Nguyên nhân là bởi vì các cập nhật chỉ phụ thuộc vào những gradient gần nhất. Một trong những giải thuật mở rộng được đưa ra để giải quyết vấn đề này là AMSGRAD. Các tác giả đã đưa ra những thí nghiệm chứng minh AMSGRAD hoạt động tốt hơn so với ADAM, nhưng trong một số thực nghiệm của nhóm, AMSGRAD chỉ có hiệu năng ngang ngửa ADAM và cả hai thuật toán đều hội tụ về cùng một giá trị.

Tài liệu tham khảo

[1] Matthew D. Zeiler, *Adadelta: an Adaptive learning rate method*, arXiv:1212.5701

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

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

[4] Sashank J.Reddi, Satyen Kate, Sanjiv Kumar, *On the convergence of adam and beyond*, ICLR 2018

[5] Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola, *Dive into Deep Learning*, Preview version at d2l.ai

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

[7] Vitaly Bushaev, *Adam a lates trends in deep learning optimization*, https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

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