



Đại học Bách Khoa - TP. Hồ Chí Minh
Khoa Khoa học và Kỹ thuật Máy Tính

THUẬT TOÁN TỐI ƯU ADAGRAD

Môn học: Cơ sở Toán cho Khoa Học Máy Tính

Lớp CH01 - HK 251 - Nhóm 1

Giáo viên hướng dẫn: TS. Nguyễn An Khương

Học viên:

Nguyễn Hoàng Long - 2570247

Lê Minh Quân - 2570307

Nguyễn Thành Đạt - 2111018

Lê Minh Huy - 2570411

Thành Phố Hồ Chí Minh, Tháng 12 Năm 2025



MỤC LỤC

- ĐẶT VĂN ĐỀ
- TIỀN ĐIỀU KIỆN
- THUẬT TOÁN ADAGRAD
- ỨNG DỤNG
- BÀI TẬP 1 - 6
- TÀI LIỆU THAM KHẢO



ĐẶT VĂN ĐỀ

Xét các bài toán với đặc trưng thừa, tức là các đặc trưng hiếm khi xuất hiện hoặc có nhiều giá trị không trong cơ sở dữ liệu. Các tham số liên quan đến chúng chỉ cập nhật rất ít lần trong suốt quá trình huấn luyện. Đối với việc áp dụng một tốc độ đọc cho toàn bộ tham số, điều này có thể dẫn đến tốc độ học với đặc trưng hiếm lại quá chậm, trong khi với đặc trưng thường gấp lại quá nhanh.

Một cách đơn giản để khắc phục là đếm số lần xuất hiện của một đặc trưng và dùng để điều chỉnh tốc độ học cho từng đặc trưng thay vì một tốc độ học cho toàn bộ. Tuy nhiên, với trường hợp các đặc trưng không hẳn là thừa, mà chỉ có gradient nhỏ. Ngay việc phân định rõ ràng rằng một gradient phải lớn như thế nào mới được coi là một quan sát cũng đã gây khó khăn.

ĐẶT VẤN ĐỀ

Adagrad được đề xuất để giải quyết vấn đề này bằng cách thay đổi bộ đếm thô bởi tổng bình phương tất cả các gradient được quan sát trước đó. Việc làm này đem lại 2 lợi ích:

- Không cần phải quyết định gradient đủ lớn để xác định một lần xuất hiện,
- Tự động thay đổi tốc độ học tùy theo gradient. Các đặc trưng với gradient lớn (thay đổi lớn) sẽ bị giảm tốc độ học đi đáng kể, trong khi các đặc trưng thừa hoặc gradient nhỏ vẫn duy trì được phần nhiều tốc độ học.

TIỀN ĐIỀU KIỆN

Xét hàm mục tiêu bậc 2 như sau:

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{c}^T \mathbf{x} + b$$

Ta có thể phân tích hệ riêng $\mathbf{Q} = \mathbf{U}^T \boldsymbol{\Lambda} \mathbf{U}$, chọn $\bar{\mathbf{x}} = \mathbf{U} \mathbf{x}$ và tương tự $\bar{\mathbf{c}} = \mathbf{U} \mathbf{c}$, hàm mục tiêu ban đầu sẽ trở thành:

$$f(\mathbf{x}) = \bar{f}(\bar{\mathbf{x}}) = \frac{1}{2} \bar{\mathbf{x}}^T \boldsymbol{\Lambda} \bar{\mathbf{x}} + \bar{\mathbf{c}}^T \bar{\mathbf{x}} + b.$$

Hàm này đạt cực tiểu tại $m = b - \frac{1}{2} \bar{\mathbf{c}}^T \mathbf{Q}^{-1} \bar{\mathbf{c}}$ với nghiệm là $\bar{\mathbf{x}}_0 = -\boldsymbol{\Lambda}^{-1} \bar{\mathbf{c}}$. Công thức này dễ tính toán hơn nhiều so với việc tính toán qua \mathbf{Q} do $\boldsymbol{\Lambda}$ là ma trận chéo chứa các trị riêng của \mathbf{Q} .

Phân tích này mở ra một hướng mới: biến đổi không gian sao cho tất cả các trị riêng đều có giá trị bằng 1. Điều này khá đơn giản trên lý thuyết: chỉ cần tính các trị riêng và vector riêng của \mathbf{Q} nhằm biến đổi bài toán từ \mathbf{x} sang $\mathbf{z} := \boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{U} \mathbf{x}$. Trong hệ toạ độ mới, $\mathbf{x}^T \mathbf{Q} \mathbf{x}$ trở thành $\|\mathbf{z}\|^2$. Nhưng hướng giải quyết này không thực tế, bởi vì việc tính trị riêng và vector riêng thường rất tốn kém.

TIỀN ĐIỀU KIỆN

Trong thực tế, ta không tính chính xác các trị riêng mà sẽ ước chừng bằng việc sử dụng các phần tử trên đường chéo và tái tỉ lệ chúng một cách tương ứng. Việc này có chi phí tính toán thấp hơn nhiều so với tính các trị riêng. Preconditioner theo cách này có dạng:

$$\tilde{\mathbf{Q}} = \text{diag}^{-\frac{1}{2}}(\mathbf{Q})\mathbf{Q}\text{diag}^{-\frac{1}{2}}(\mathbf{Q}).$$

Từ đây, có thể thấy $\tilde{\mathbf{Q}}_{ij} = \mathbf{Q}_{ij}/\sqrt{\mathbf{Q}_{ii}\mathbf{Q}_{jj}}$ và $\tilde{\mathbf{Q}}_{ii} = 1$ với mọi i . Trong đa số các trường hợp, cách làm này sẽ đơn giản hóa đáng kể hệ số điều kiện. Nói cách khác, cách làm này giúp ta ước chừng trị riêng, vector riêng của \mathbf{Q} mà không tốn quá nhiều chi phí.

Tuy nhiên, khi sử dụng ma trận Hessian để làm preconditioner như phương pháp Newton, ta lại có vấn đề: trong học sâu, thường không tinh được ngay cả đạo hàm bậc hai của hàm mục tiêu bởi yêu cầu không gian và độ phức tạp lớn. Sự khéo léo của Adagrad nằm ở việc sử dụng một biến đại diện ước chừng để tính toán đường chéo của ma trận Hessian một cách hiệu quả và đơn giản—đó là độ lớn của chính gradient.

THUẬT TOÁN

Ta formal hoá lại phần thuật toán Adagrad. Sử dụng biến s_t để cộng dồn bình phương các gradient trong quá khứ như sau (12.7.5):

$$\mathbf{g}_t = \nabla_{\mathbf{w}} l(y_t, f(\mathbf{x}_t, \mathbf{w}))$$

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

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \mathbf{g}_t$$

Ở đây tất cả phép toán đều được áp dụng theo từng toạ độ (coordinate-wise).

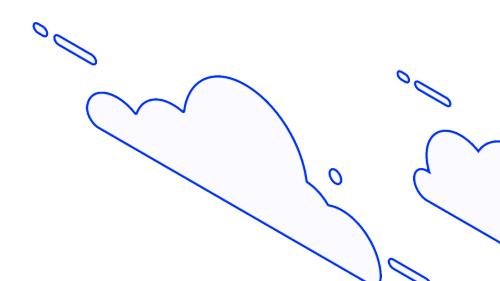
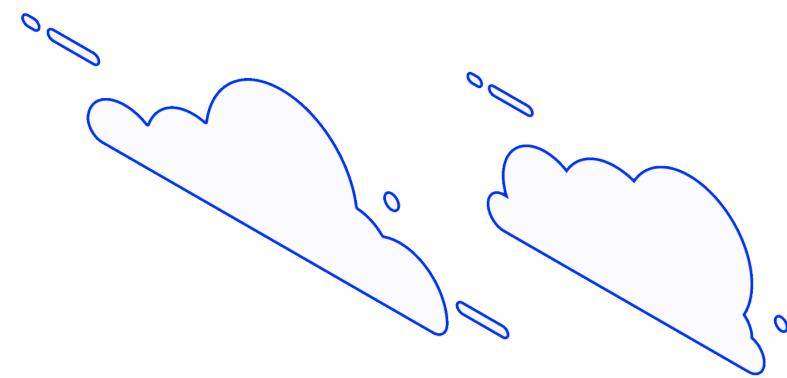
Nếu \mathbf{v} là một vector thì:

- \mathbf{v}^2 là vector có phần tử thứ i bằng v_i^2
- $\frac{1}{\sqrt{\mathbf{v}}}$ có phần tử thứ i bằng $\frac{1}{\sqrt{v_i}}$
- tích theo toạ độ $\mathbf{u} \cdot \mathbf{v}$ có phần tử thứ i bằng $u_i v_i$

Trong công thức cập nhật, η là learning rate ban đầu (toàn cục) và ϵ là một hằng số rất nhỏ dùng để tránh chia cho 0. Ta khởi tạo $\mathbf{s}_0 = \mathbf{0}$.

ỨNG DỤNG

Adagrad được dùng để làm
gì? Giải bài toán nào?





ỨNG DỤNG



→ Tại sao lại dùng Adagrad?



- Tự động điều chỉnh tốc độ học.
- Xử lý tốt dữ liệu thưa (sparse) và đa chiều (high-dimensional).
- Hiệu quả trong bối cảnh dữ liệu luồng.
- Giảm nhu cầu tinh chỉnh siêu tham số.



ỨNG DỤNG

Học dữ liệu theo thời gian thực (Online learning)

Vì sao AdaGrad phù hợp:

- Điều chỉnh learning rate theo từng tham số → thích hợp dữ liệu streaming.
- Xử lý tốt dữ liệu thay đổi liên tục trong môi trường online.

Điểm mạnh:

- Giảm regret tốt hơn các phương pháp SGD truyền thống.
- Ổn định hơn khi dữ liệu xuất hiện theo thời gian thực.

Dẫn chứng:

- Bài báo “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization” (Duchi et al.).
→ AdaGrad cho hiệu suất vượt trội và regret guarantee tốt hơn SGD không thích ứng.

ỨNG DỤNG

Hệ thống gợi ý thời gian thực (Real-time recommendation system)

Vì sao AdaGrad phù hợp:

- Dữ liệu tương tác người dùng rất thưa → nhiều item không được click
- AdaGrad ưu tiên học tốt hơn trên các “rare features”
- Learning rate thích ứng giúp mô hình cập nhật nhanh khi hành vi người dùng thay đổi

Ứng dụng:

- Recommendation theo thời gian thực cho phim, truyện, nhạc, sách,...

Dẫn chứng:

- Bài báo “Movie Recommender System on Twitter Using Weighted Hybrid Filtering and GRU” (Valentino and Setiawan).
→ So sánh AdaGrad với Adam, Nadam, Adadelta, SGD... để tìm thuật toán tối ưu

ỨNG DỤNG

Phân tích dữ liệu theo thời gian (Time-series analytics)

Khi nào nên dùng AdaGrad

- Xuất hiện nhiều sự kiện hiếm hoặc bất thường (rare spikes/outliers)
- Cần điều chỉnh learning rate tự động cho từng đặc trưng theo thời gian

Ứng dụng:

- Dự đoán lượng truy cập ngày lễ.
- Phát hiện thời tiết bất thường.
- Lọc nhiễu dữ liệu luồng.

Dẫn chứng

- Bài báo “Improving Multiple Time Series Forecasting with Data Stream Mining Algorithms” (Mochinski et al.).
→ Kết hợp AdaGrad + ARIMA để chứng minh sự kết hợp hiệu quả thông qua một vài bài toán dạng time-series.

ỨNG DỤNG

Nhận diện hình ảnh (Image Recognition)

Vai trò của AdaGrad:

- Dùng trong các mô hình trước khi CNN ra đời.
- Không phổ biến bằng Adam, RMSProp nhưng vẫn hiệu quả trong bối cảnh đặc thù (dữ liệu thưa, đa chiều).
- Các biến thể Adagrad giúp khắc phục vấn đề tốc độ học giảm quá nhanh

Hạn chế của AdaGrad gốc:

- Tốc độ học giảm dần quá nhanh → khó train deep networks dài hạn.

Dẫn chứng:

- Bài báo “Learning Rate Re-Scheduling for Adagrad in Training Deep Neural Networks” để xuất biến thể AdagradW.
→ Cải thiện bằng cách sử dụng kỹ thuật Learning Rate Rescheduling để cải thiện vấn đề tốc độ học giảm dần quá nhanh.

BÀI TẬP

Trình bày các bài tập 1-6
Cách giải chi tiết và giải thích

BÀI TẬP

Câu 1. Prove that for an orthogonal matrix \mathbf{U} and a vector \mathbf{c} the following holds: $\|\mathbf{c} - \delta\|_2 = \|\mathbf{U}\mathbf{c} - \mathbf{U}\delta\|_2$. Why does this mean that the magnitude of perturbations does not change after an orthogonal change of variables?

Ta có:

$$\begin{aligned}\|\mathbf{U}\mathbf{c} - \mathbf{U}\delta\|_2^2 &= (\mathbf{U}\mathbf{c} - \mathbf{U}\delta)^\top (\mathbf{U}\mathbf{c} - \mathbf{U}\delta) \\ &= (\mathbf{U}(\mathbf{c} - \delta))^\top (\mathbf{U}(\mathbf{c} - \delta)) \\ &= (\mathbf{c} - \delta)^\top \mathbf{U}^\top \mathbf{U}(\mathbf{c} - \delta)\end{aligned}$$

Vì \mathbf{U} là ma trận trực giao nên $\mathbf{U}^\top \mathbf{U} = \mathbf{I}$

$$\begin{aligned}\Rightarrow \|\mathbf{U}\mathbf{c} - \mathbf{U}\delta\|_2^2 &= (\mathbf{c} - \delta)^\top \mathbf{I}(\mathbf{c} - \delta) \\ &= (\mathbf{c} - \delta)^\top (\mathbf{c} - \delta) \\ &= \|\mathbf{c} - \delta\|_2^2\end{aligned}$$

Lấy căn bậc 2 hai vế ta được kết quả cần chứng minh:

$$\|\mathbf{c} - \delta\|_2 = \|\mathbf{U}\mathbf{c} - \mathbf{U}\delta\|_2$$

BÀI TẬP

Từ kết quả trên, ta rút ra một số kết luận:

- Khi biến đổi trực giao nhiễu $\mathbf{c} - \delta$ (perturbation), độ lớn của nó sẽ không thay đổi
- Việc đặt $\bar{\mathbf{x}} = \mathbf{U}\mathbf{x}$ và $\bar{\mathbf{c}} = \mathbf{U}\mathbf{c}$ để chéo hóa ma trận \mathbf{Q} ở công thức 12.7.1 trong D2L sẽ đảm bảo:
 - Tốc độ hội tụ không bị bóp méo
 - Tốc độ học η không cần điều chỉnh lại
 - Phân tích hội tụ trong không gian mới có ý nghĩa thực sự trong không gian cũ
 - Không làm thay đổi độ lớn bước nhảy của gradient hay nhiễu

BÀI TẬP

Câu 2. Try out Adagrad for $f(\mathbf{x}) = 0.1x_1^2 + 2x_2^2$ and also for the objective function rotated by 45° , i.e.,

$f(\mathbf{x}) = 0.1(x_1 + x_2)^2 + 2(x_1 - x_2)^2$. Does it behave differently?

Adagrad cho $f(\mathbf{x}) = 0.1x_1^2 + 2x_2^2$

Ta xét hàm:

$$f(x_1, x_2) = 0.1x_1^2 + 2x_2^2$$

và dùng Adagrad để tối ưu, với:

- Khởi tạo: $\mathbf{x}^{(0)} = (2, 1)$
- Bộ nhớ: $\mathbf{s}^{(0)} = (0, 0)$
- Learning rate: $\eta = 0.4$

BÀI TẬP

1. Gradient của hàm

Đạo hàm riêng:

$$\frac{\partial f}{\partial x_1} = 0.2x_1 \quad \frac{\partial f}{\partial x_2} = 4x_2$$

Nên gradient tại bước (t):

$$\nabla f(x^{(t)}) = \begin{bmatrix} g_1^{(t)} \\ g_2^{(t)} \end{bmatrix} = \begin{bmatrix} 0.2 x_1^{(t)} \\ 4 x_2^{(t)} \end{bmatrix}$$

2. Công thức Adagrad trong 2 chiều

Kí hiệu:

- $x^{(t)} = (x_1^{(t)}, x_2^{(t)})$
- $g^{(t)} = (g_1^{(t)}, g_2^{(t)}) = \nabla f(x^{(t)})$
- $s^{(t)} = (s_1^{(t)}, s_2^{(t)})$

Cập nhật bộ nhớ

$$s_i^{(t+1)} = s_i^{(t)} + (g_i^{(t)})^2, \quad i = 1, 2$$

Cập nhật tham số

$$x_i^{(t+1)} = x_i^{(t)} - \eta \frac{g_i^{(t)}}{\sqrt{s_i^{(t+1)}} + \varepsilon}, \quad i = 1, 2$$

Trong ví dụ này ta coi $\varepsilon \approx 0$ cho gọn.

BÀI TẬP

3. Tính toán

Bước 0

Khởi tạo: $x^{(0)} = (2, 1)$, $s^{(0)} = (0, 0)$

Giá trị hàm:

$$f(x^{(0)}) = 0.1 \cdot 2^2 + 2 \cdot 1^2 = 0.4 + 2 = 2.4$$

Gradient:

$$g_1^{(0)} = 0.2 \cdot 2 = 0.4, \quad g_2^{(0)} = 4 \cdot 1 = 4$$

Cập nhật bộ nhớ:

$$s_1^{(1)} = 0 + 0.4^2 = 0.16, \quad s_2^{(1)} = 0 + 4^2 = 16$$

Cập nhật tham số:

$$x_1^{(1)} = 2 - 0.4 \cdot \frac{0.4}{\sqrt{0.16}} = 2 - 0.4 = 1.6$$

$$x_2^{(1)} = 1 - 0.4 \cdot \frac{4}{\sqrt{16}} = 1 - 0.4 = 0.6$$

Vậy:

$$x^{(1)} = (1.6, 0.6)$$

và

$$f(x^{(1)}) = 0.1 \cdot 1.6^2 + 2 \cdot 0.6^2 = 0.256 + 0.72 = 0.976$$

BÀI TẬP

Bước 1

Gradient tại $x^{(1)}$:

$$g_1^{(1)} = 0.2 \cdot 1.6 = 0.32, \quad g_2^{(1)} = 4 \cdot 0.6 = 2.4$$

Cập nhật bộ nhớ:

$$s_1^{(2)} = 0.16 + 0.32^2 = 0.2624$$

$$s_2^{(2)} = 16 + 2.4^2 = 21.76$$

Cập nhật tham số (lấy gần đúng):

$$x_1^{(2)} = 1.6 - 0.4 \cdot \frac{0.32}{\sqrt{0.2624}} \approx 1.35$$

$$x_2^{(2)} = 0.6 - 0.4 \cdot \frac{2.4}{\sqrt{21.76}} \approx 0.394$$

Nên

$$x^{(2)} \approx (1.35, 0.394)$$

và

$$f(x^{(2)}) \approx 0.493$$

BÀI TẬP

Bước 2

Gradient tại $x^{(2)}$:

$$g_1^{(2)} = 0.2 \cdot 1.35 = 0.27, \quad g_2^{(2)} = 4 \cdot 0.394 \approx 1.576$$

Cập nhật bộ nhớ:

$$s_1^{(3)} = s_1^{(2)} + (g_1^{(2)})^2 = 0.2624 + 0.27^2 = 0.2624 + 0.0729 \approx 0.3353$$

$$s_2^{(3)} = s_2^{(2)} + (g_2^{(2)})^2 = 21.76 + 1.576^2 \approx 21.76 + 2.4838 \approx 24.2438$$

Cập nhật tham số:

$$x_1^{(3)} = x_1^{(2)} - \eta \frac{g_1^{(2)}}{\sqrt{s_1^{(3)}}} = 1.35 - 0.4 \cdot \frac{0.27}{\sqrt{0.3353}} \approx 1.1635 \approx 1.164$$

$$x_2^{(3)} = x_2^{(2)} - \eta \frac{g_2^{(2)}}{\sqrt{s_2^{(3)}}} = 0.394 - 0.4 \cdot \frac{1.576}{\sqrt{24.2438}} \approx 0.266$$

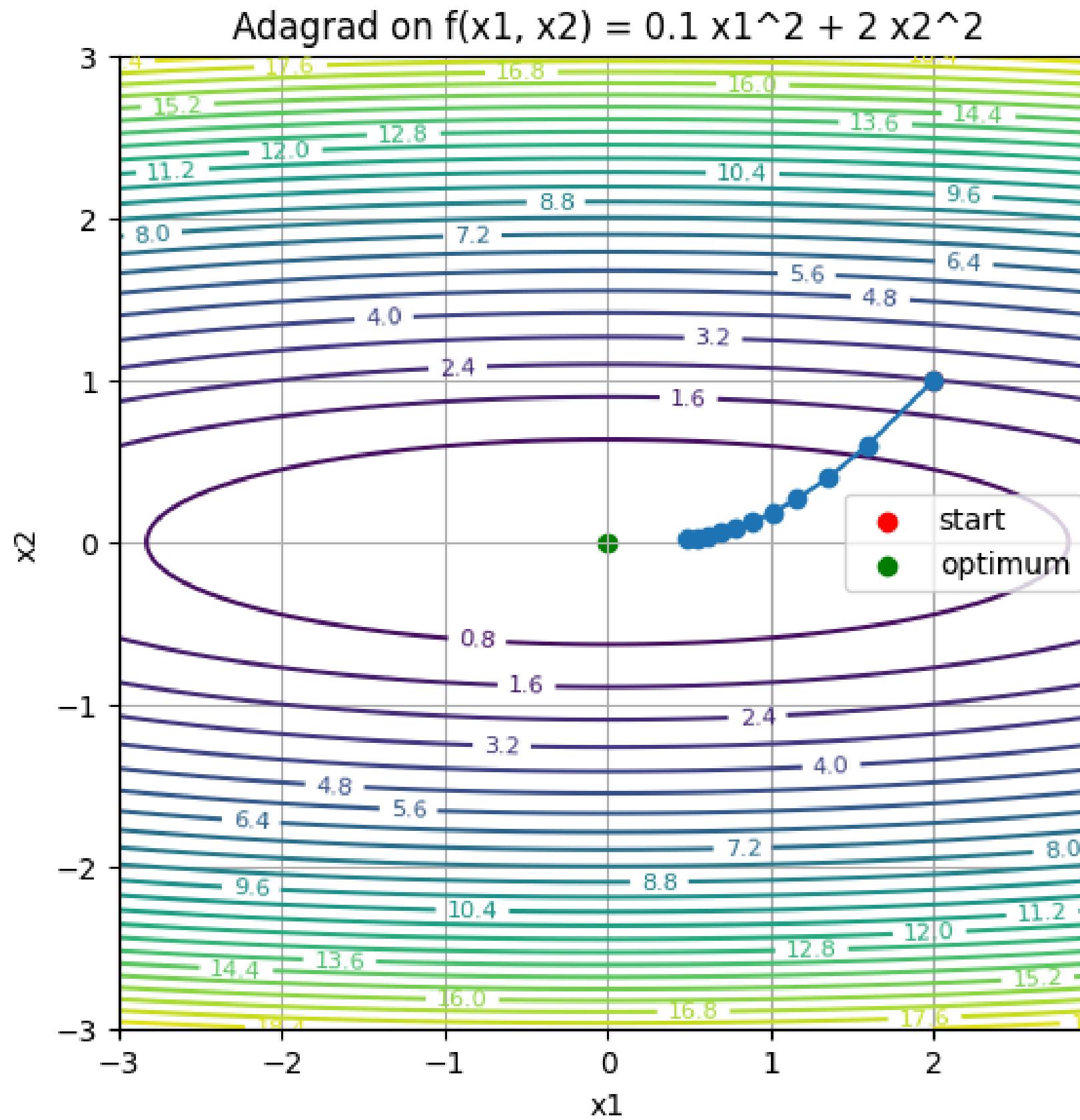
Nên

$$x^{(3)} \approx (1.164, 0.266)$$

và

$$f(x^{(3)}) = 0.1(x_1^{(3)})^2 + 2(x_2^{(3)})^2 \approx 0.277.$$

BÀI TẬP



Quan sát biểu đồ contour ta thấy:

- Đường đi quỹ đạo của Adagrad xuất phát từ điểm (2,1) tiến dần về nghiệm tối ưu (0,0).
- Bước đi ban đầu dài, sau đó ngắn lại khi tích lũy bình phương gradient tăng lên \rightarrow learning rate hiệu dụng tự động giảm.
- Nhờ điều chỉnh riêng cho từng chiều, Adagrad hội tụ ổn định trên hàm này.

BÀI TẬP

Adagrad cho hàm xoay 45° $f(x) = 0.1(x_1 + x_2)^2 + 2(x_1 - x_2)^2$

Ta xét hàm

$$f(x_1, x_2) = 0.1(x_1 + x_2)^2 + 2(x_1 - x_2)^2$$

và dùng Adagrad để tối ưu, với cùng cấu hình như trước:

- Khởi tạo: $x^{(0)} = (2, 1)$
- Bộ nhớ: $s^{(0)} = (0, 0)$
- Learning rate: $\eta = 0.4$

1. Gradient của hàm xoay

Đặt

$$u = x_1 + x_2, \quad v = x_1 - x_2, \quad f = 0.1u^2 + 2v^2.$$

Đạo hàm riêng:

$$\frac{\partial f}{\partial x_1} = 0.1 \cdot 2u \cdot \frac{\partial u}{\partial x_1} + 2 \cdot 2v \cdot \frac{\partial v}{\partial x_1} = 0.2(x_1 + x_2) + 4(x_1 - x_2),$$

$$\frac{\partial f}{\partial x_2} = 0.1 \cdot 2u \cdot \frac{\partial u}{\partial x_2} + 2 \cdot 2v \cdot \frac{\partial v}{\partial x_2} = 0.2(x_1 + x_2) - 4(x_1 - x_2).$$

Nên gradient tại bước (t):

$$\nabla f(x^{(t)}) = \begin{bmatrix} g_1^{(t)} \\ g_2^{(t)} \end{bmatrix} = \begin{bmatrix} 0.2(x_1^{(t)} + x_2^{(t)}) + 4(x_1^{(t)} - x_2^{(t)}) \\ 0.2(x_1^{(t)} + x_2^{(t)}) - 4(x_1^{(t)} - x_2^{(t)}) \end{bmatrix}.$$

BÀI TẬP

2. Tính toán

Bước 0

Khởi tạo: $x^{(0)} = (2, 1)$ và $s^{(0)} = (0, 0)$

Giá trị hàm:

$$f(x^{(0)}) = 0.1(2+1)^2 + 2(2-1)^2 = 0.1 \cdot 9 + 2 \cdot 1 = 2.9$$

Gradient tại $x^{(0)}$:

$$\begin{aligned}g_1^{(0)} &= 0.2(2+1) + 4(2-1) = 0.6 + 4 = 4.6, \\g_2^{(0)} &= 0.2(2+1) - 4(2-1) = 0.6 - 4 = -3.4.\end{aligned}$$

Cập nhật bộ nhớ:

$$s_1^{(1)} = 0 + 4.6^2 = 21.16, \quad s_2^{(1)} = 0 + (-3.4)^2 = 11.56.$$

Cập nhật tham số:

$$x_1^{(1)} = 2 - 0.4 \cdot \frac{4.6}{\sqrt{21.16}} = 2 - 0.4 = 1.6,$$

$$x_2^{(1)} = 1 - 0.4 \cdot \frac{-3.4}{\sqrt{11.56}} = 1 + 0.4 = 1.4.$$

Vậy

$$x^{(1)} = (1.6, 1.4),$$

và

$$f(x^{(1)}) = 0.1(1.6+1.4)^2 + 2(1.6-1.4)^2 = 0.1 \cdot 3^2 + 2 \cdot 0.2^2 = 0.9 + 0.08 = 0.98.$$

BÀI TẬP

Bước 2

Lúc này ta đang có xấp xỉ: $x^{(2)} \approx (1.48, 1.42)$, $s^{(2)} = (23.12, 11.60)$.

Gradient tại $x^{(2)}$:

Ta có

$$x_1^{(2)} + x_2^{(2)} = 1.48 + 1.42 = 2.90, \quad x_1^{(2)} - x_2^{(2)} = 1.48 - 1.42 = 0.06.$$

Do đó

$$\begin{aligned} g_1^{(2)} &= 0.2 \cdot 2.90 + 4 \cdot 0.06 = 0.58 + 0.24 = 0.82, \\ g_2^{(2)} &= 0.2 \cdot 2.90 - 4 \cdot 0.06 = 0.58 - 0.24 = 0.34. \end{aligned}$$

Cập nhật bộ nhớ:

$$\begin{aligned} s_1^{(3)} &= s_1^{(2)} + (g_1^{(2)})^2 = 23.12 + 0.82^2 = 23.12 + 0.6724 \approx 23.79, \\ s_2^{(3)} &= s_2^{(2)} + (g_2^{(2)})^2 = 11.60 + 0.34^2 = 11.60 + 0.1156 \approx 11.72. \end{aligned}$$

Cập nhật tham số:

$$\begin{aligned} x_1^{(3)} &= x_1^{(2)} - \eta \frac{g_1^{(2)}}{\sqrt{s_1^{(3)}}} = 1.48 - 0.4 \cdot \frac{0.82}{\sqrt{23.79}} \approx 1.41, \\ x_2^{(3)} &= x_2^{(2)} - \eta \frac{g_2^{(2)}}{\sqrt{s_2^{(3)}}} = 1.42 - 0.4 \cdot \frac{0.34}{\sqrt{11.72}} \approx 1.38. \end{aligned}$$

Nên

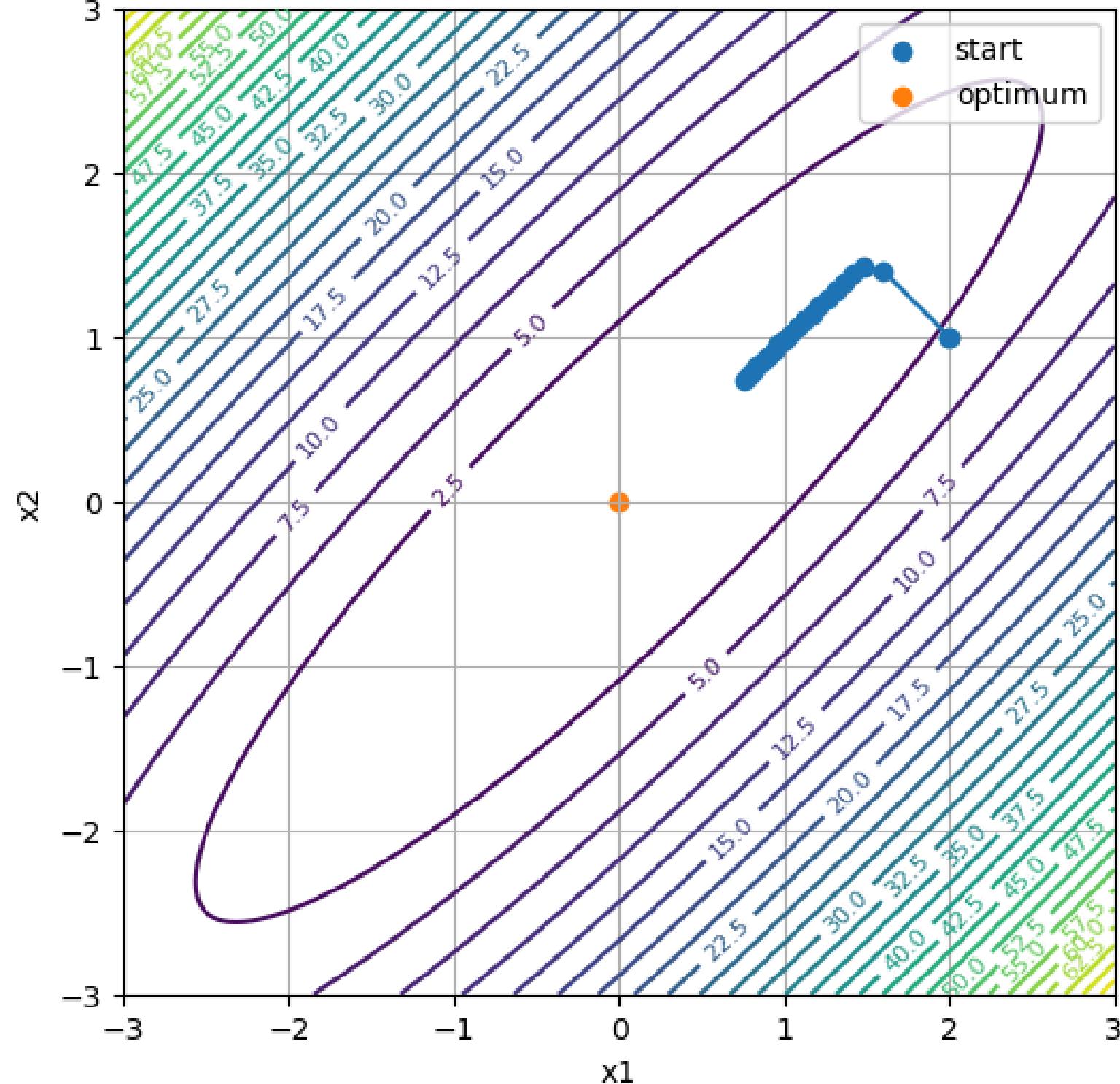
$$x^{(3)} \approx (1.41, 1.38),$$

và giá trị hàm tại bước này:

$$f(x^{(3)}) = 0.1(x_1^{(3)} + x_2^{(3)})^2 + 2(x_1^{(3)} - x_2^{(3)})^2 \approx 0.78.$$

BÀI TẬP

Adagrad on rotated $f(x_1, x_2) = 0.1 (x_1 + x_2)^2 + 2 (x_1 - x_2)^2$



- Ở hàm đã xoay, các hình elipse không còn song song với trục x_1 và x_2 nữa.
- Adagrad vẫn tiến về nghiệm tối ưu, nhưng quỹ đạo bị xiên và chậm hơn, bước đi ngắn dần, cần nhiều step mới có thể tiếng về nghiệm tố ưu hơn so với hàm gốc.
- Điều này cho thấy, khi trục tọa độ không thẳng hàng với hướng cong mạnh/yếu của hàm, Adagrad sẽ hội tụ khó hơn so với trường hợp chuẩn tắc.

BÀI TẬP

Câu 3. Prove [Gershgorin's circle theorem](#) which states that eigenvalues λ_i of a matrix \mathbf{M} satisfy $|\lambda_i - \mathbf{M}_{jj}| \leq \sum_{k \neq j} |\mathbf{M}_{jk}|$ for at least one choice of j .

Gọi vector $\mathbf{x}_i = (\mathbf{x}_{ik})$ là vector riêng tương ứng với trị riêng λ_i . Lấy j sao cho \mathbf{x}_{ij} là phần tử có trị tuyệt đối lớn nhất trong \mathbf{x}_i . Theo định nghĩa của trị riêng, vector riêng, ta có: $\mathbf{Mx}_i = \lambda_i \mathbf{x}_i$. Xét phần tử thứ m của phương trình, ta có:

$$\sum_k \mathbf{M}_{jk} \mathbf{x}_{ik} = \lambda_i \mathbf{x}_{ij}.$$

Chuyển về $\mathbf{M}_{jj} \mathbf{x}_{ij}$, ta được:

$$\sum_{k \neq j} \mathbf{M}_{jk} \mathbf{x}_{ik} = \lambda_i \mathbf{x}_{ij} - \mathbf{M}_{jj} \mathbf{x}_{ij} = (\lambda_i - \mathbf{M}_{jj}) \mathbf{x}_{ij}.$$

Do đó:

$$|\lambda_i - \mathbf{M}_{jj}| = \left| \sum_{k \neq j} \frac{\mathbf{M}_{jk} \mathbf{x}_{ik}}{\mathbf{x}_{ij}} \right| \leq \sum_{k \neq j} \left| \mathbf{M}_{jk} \frac{\mathbf{x}_{ik}}{\mathbf{x}_{ij}} \right| = \sum_{k \neq j} |\mathbf{M}_{jk}| \left| \frac{\mathbf{x}_{ik}}{\mathbf{x}_{ij}} \right| \leq \sum_{k \neq j} |\mathbf{M}_{jk}|$$

Dấu bé hơn hoặc bằng đúng là vì ta đã chọn j sao cho \mathbf{x}_{ij} có trị tuyệt đối lớn nhất, do đó $\left| \frac{\mathbf{x}_{ik}}{\mathbf{x}_{ij}} \right| \leq 1 \forall k$. Ta được điều phải chứng minh.

BÀI TẬP

Câu 4. What does Gerschgorin's theorem tell us about the eigenvalues of the diagonally preconditioned matrix

$$\text{diag}^{-\frac{1}{2}}(\mathbf{M})\mathbf{M}\text{diag}^{-\frac{1}{2}}(\mathbf{M})?$$

Gọi $\tilde{\mathbf{M}} = \text{diag}^{-\frac{1}{2}}(\mathbf{M})\mathbf{M}\text{diag}^{-\frac{1}{2}}(\mathbf{M})$. Theo kết quả từ phần preconditioning, ta có được $\tilde{\mathbf{M}}_{ij} = \mathbf{M}_{ij}/\sqrt{\mathbf{M}_{ii}\mathbf{M}_{jj}}$ và $\tilde{\mathbf{M}}_{ii} = 1$. Thay vào định lý Gerchgorin, ta có được với λ_i là các trị riêng của $\tilde{\mathbf{M}}$, tồn tại j thoả mãn:

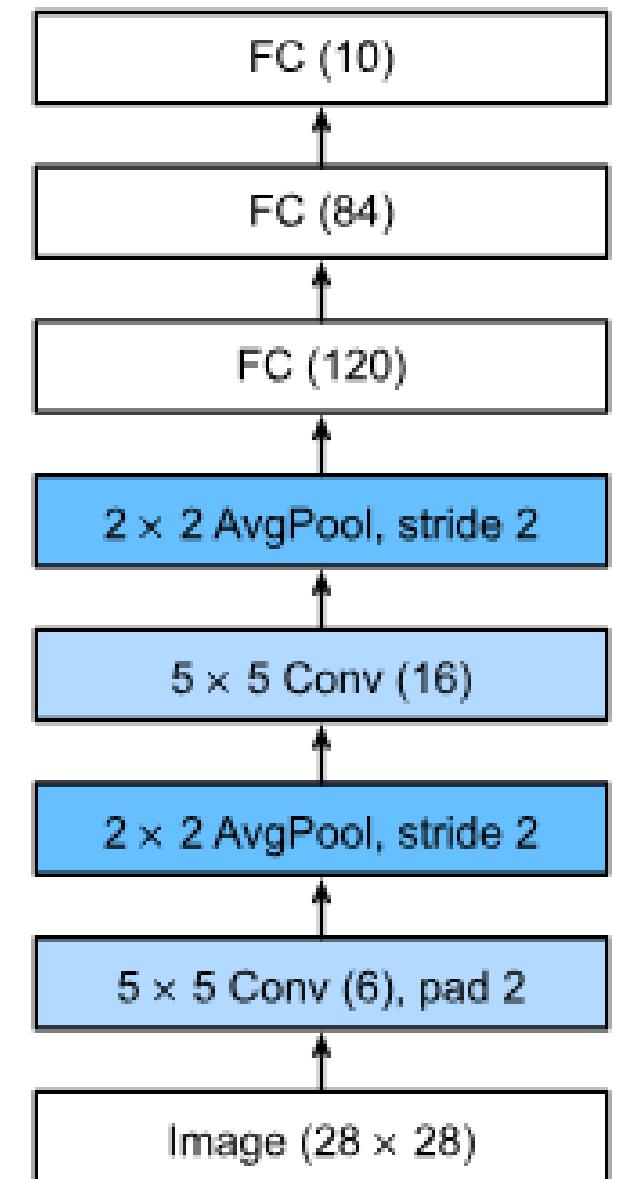
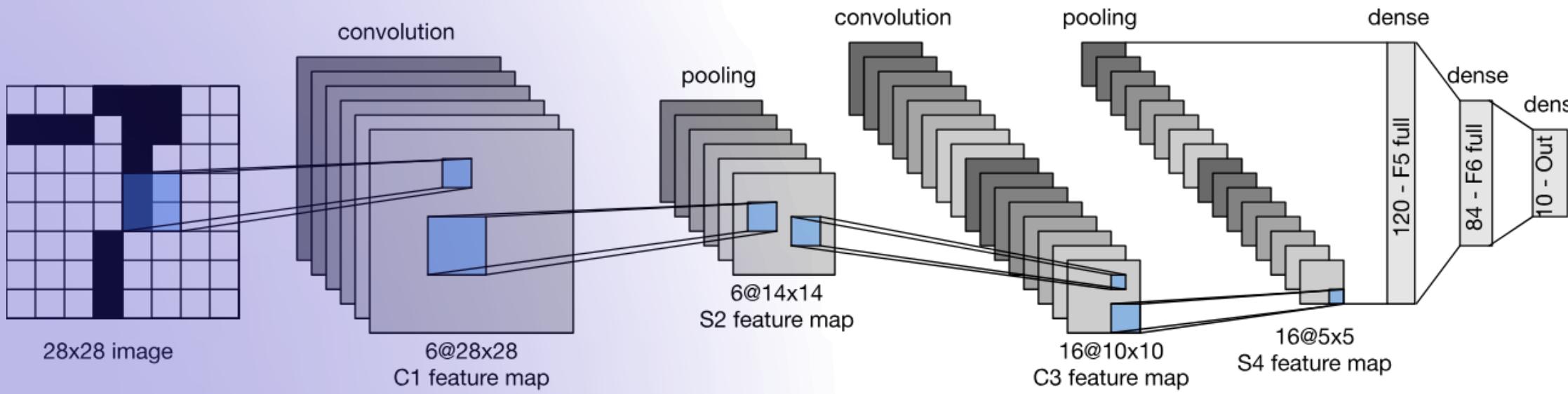
$$|\lambda_i - \tilde{\mathbf{M}}_{jj}| = |\lambda_i - 1| \leq \sum_{k \neq j} |\tilde{\mathbf{M}}_{jk}|$$

Điều này có nghĩa là toàn bộ trị riêng của ma trận $\tilde{\mathbf{M}}$ đều nằm trong khoảng lân cận (đường tròn) xung quanh 1. Do đó, ta có thể sử dụng 1 là giá trị ước chừng cho tất cả trị riêng của $\tilde{\mathbf{M}}$ - theo ý tưởng biến đổi không gian sao cho tất cả các trị riêng đều có giá trị bằng 1.

BÀI TẬP

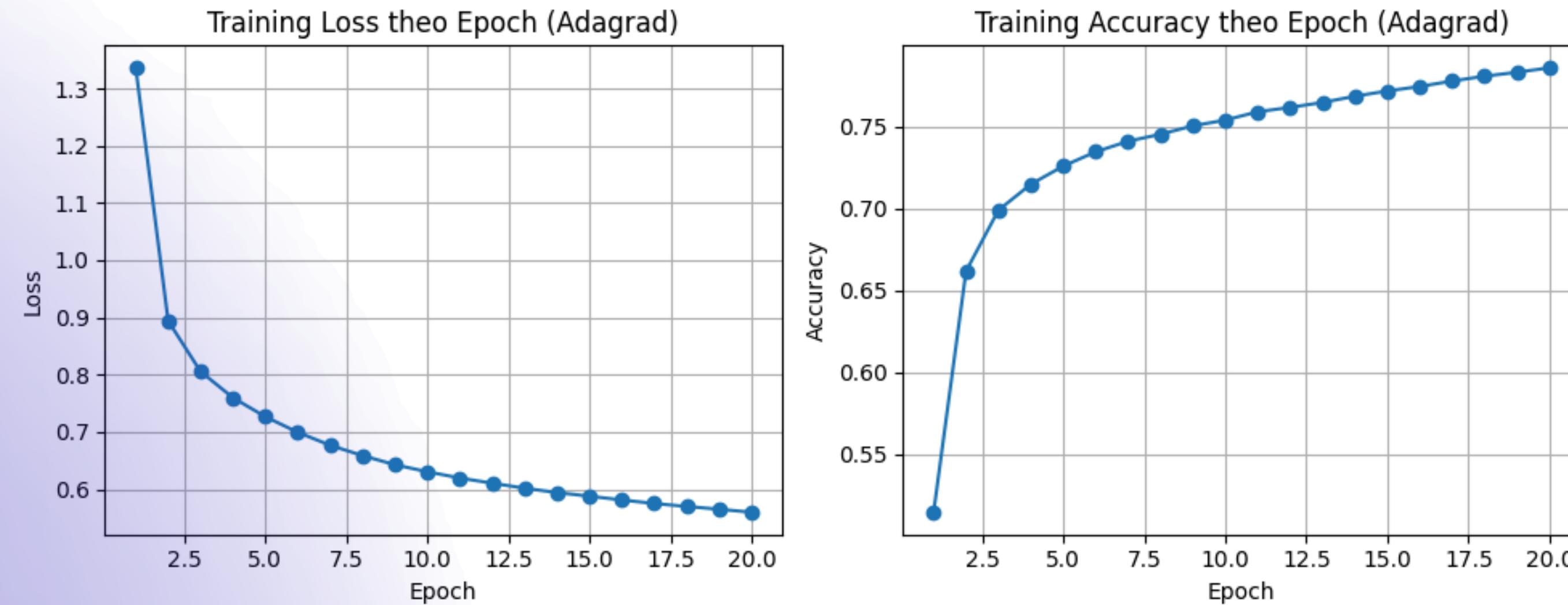
Câu 5. Try out Adagrad for a proper deep network,
such as Section 7.6 when applied to Fashion-MNIST.

Kiến trúc của mạng LeNet ở Section 7.6, áp dụng cho tập dữ liệu
Fashion-MNIST với optimizer là Adagrad



BÀI TẬP

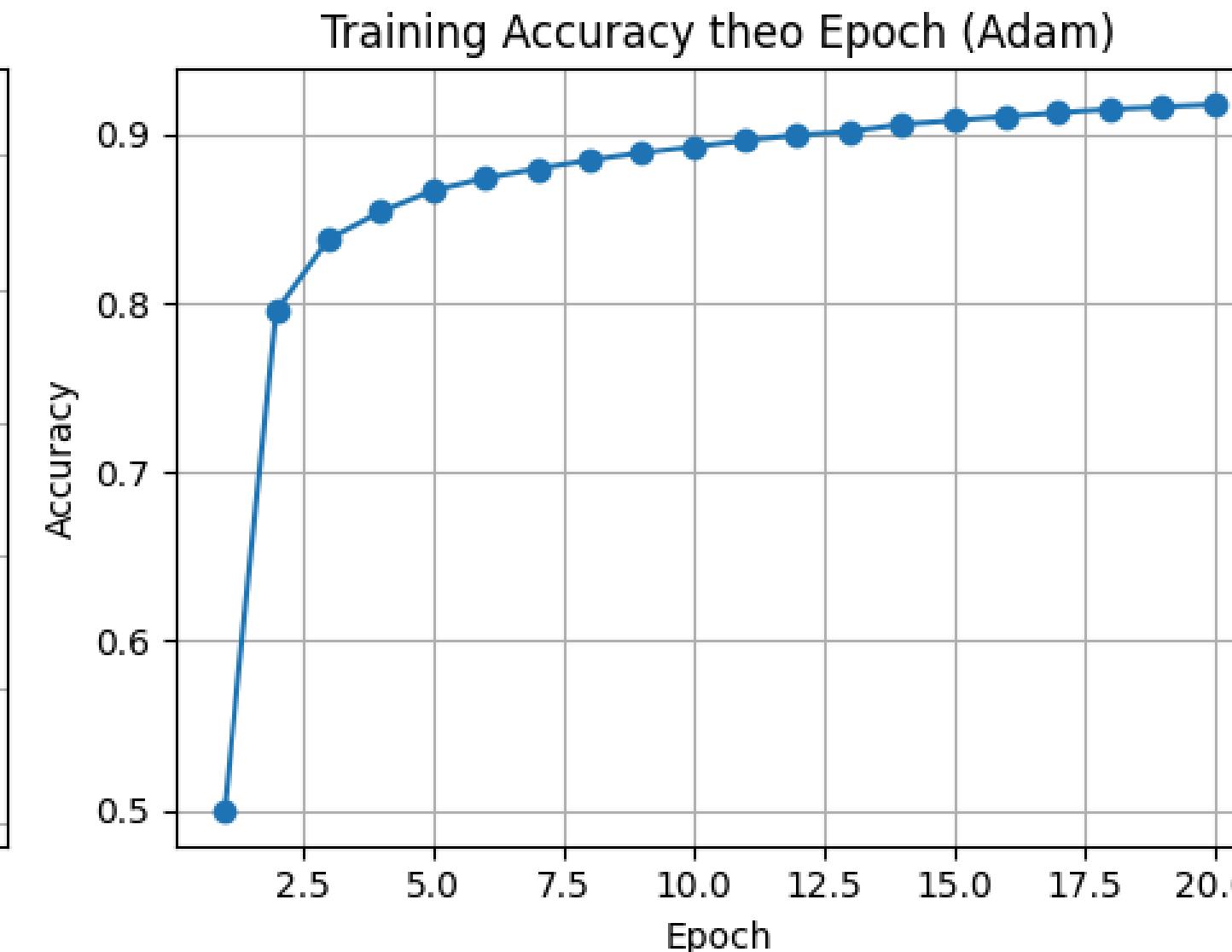
Training với Adagrad optimizer Self-Implement



- Có thể thấy với Adagrad optimizer thì loss giảm từ 1.33 xuống 0.56 sau 20 epochs, giảm đều, không bị nhảy loạn, accuracy từ 51% tăng lên 78.6%, tăng đều không bị tụt lại ở cuối.
- Tuy nhiên quan sát biểu đồ cũng như log có thể thấy epoch 1 -> 3 loss rơi từ rất nhanh từ 1.33 -> 0.8, đúng với lý thuyết Adagrad: lúc đầu G_t còn nhỏ nên bước nhảy hiệu dụng khá lớn, mô hình cải thiện rất nhanh.
- Về sau từ epoch 10 trở đi, loss chỉ giảm rất từ (0.63 -> 0.56), lý do là lúc này accumulator G_t ngày càng lớn, nên learning rate hiệu dụng bị giảm mạnh, khiến bước cập nhật nhỏ lại, hội tụ chậm hơn

BÀI TẬP

Training với Adam optimizer gọi từ Torch



- Để so sánh mức độ mạnh yếu của Adagrad so với những optimizer khác thì nhóm chúng em có chạy thử model LeNet đã tạo ở trên với một optimizer khác là Adam.
- Quan sát biểu đồ Loss và Accuracy có thể thấy, hiệu quả tối ưu hóa cao hơn. Loss giảm từ khoảng 1.28 → xuống 0.22 sau 20 epochs, còn Accuracy tăng từ 50% lên 91.8%.
- Ngay từ những epoch đầu mô hình đã học rất nhanh, sau đó vẫn tiếp tục tăng đều và chậm dần nhưng vẫn nhanh hơn Adagrad và đến cuối 20 epoch vẫn còn cải thiện nhẹ.

BÀI TẬP

Câu 6. How would you need to modify Adagrad to achieve a less aggressive decay in learning rate?

Ta có công thức của Adagrad như sau:

$$\begin{aligned}\mathbf{g}_t &= \partial_{\mathbf{w}} l(y_t, f(\mathbf{x}_t, \mathbf{w})), \\ \mathbf{s}_t &= \mathbf{s}_{t-1} + \mathbf{g}_t^2, \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \cdot \mathbf{g}_t.\end{aligned}$$

- η phân rã với tốc độ $\mathcal{O}(t^{-\frac{1}{2}})$
 - Có thể phù hợp với các bài toán lồi, nhưng sẽ không lý tưởng trong các vấn đề học sâu
- Cần phải giảm tốc độ học chậm hơn nữa

BÀI TẬP

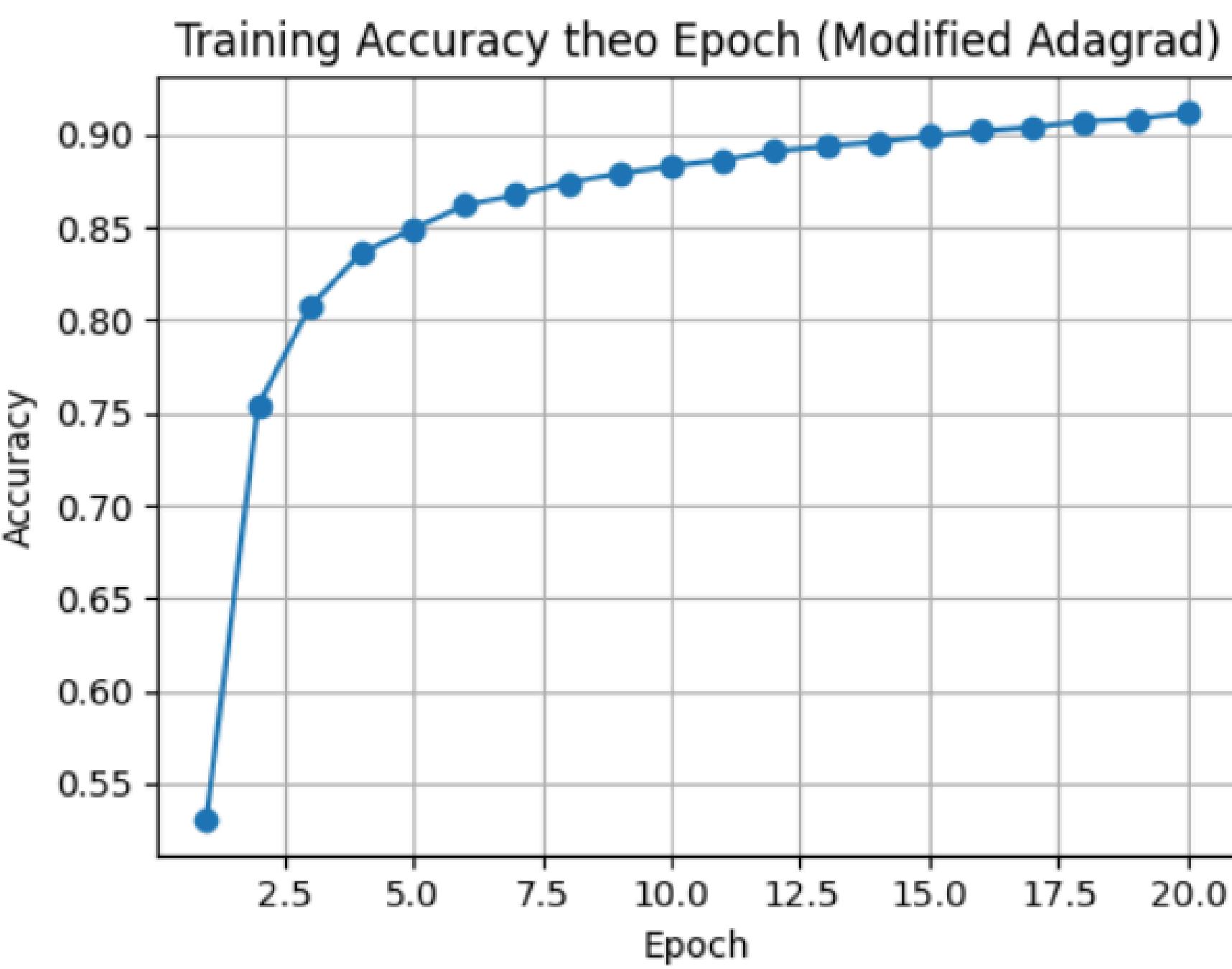
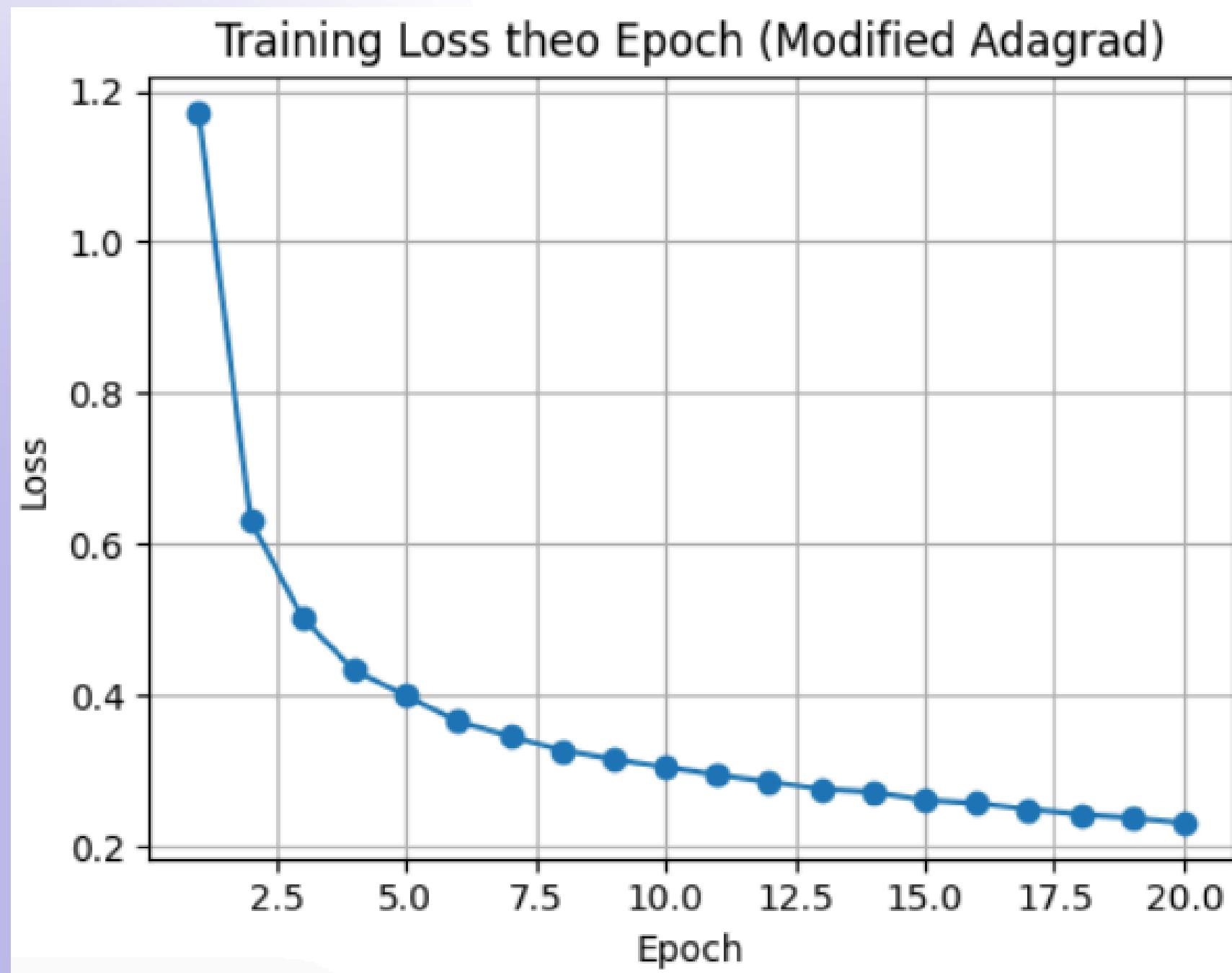
6.1. Sử dụng trung bình \mathbf{s}_t/t

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t/t + \epsilon}} \cdot \mathbf{g}_t.$$

- Tính trung bình \mathbf{s}_t theo thời gian
- $\mathcal{O}(t^0) \rightarrow \mathcal{O}(t^{-\frac{1}{2}})$
- Nếu \mathbf{g}_t có phân phối ổn định: $\frac{1}{t} \sum_{i=1}^t \mathbf{g}_i^2 \xrightarrow[t \rightarrow \infty]{} \mathbb{E}[\mathbf{g}^2] \rightarrow \mathbf{s}_t/t$ sẽ tiến dần về độ lớn trung bình của gradient theo thời gian
 - Nhưng nhược điểm của phương pháp này là sẽ tốn rất nhiều thời gian để ổn định nếu như gradient ban đầu lớn

BÀI TẬP

Training với Modified Adagrad Self-Implement



BÀI TẬP

6.2. Sử dụng thuật toán RMSProp - một biến thể của Adagrad

Tham khảo phần 12.8.1 trong D2L, thuật toán sử dụng leaky average để tính \mathbf{s}_t như sau:

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

với tham số $\gamma > 0$

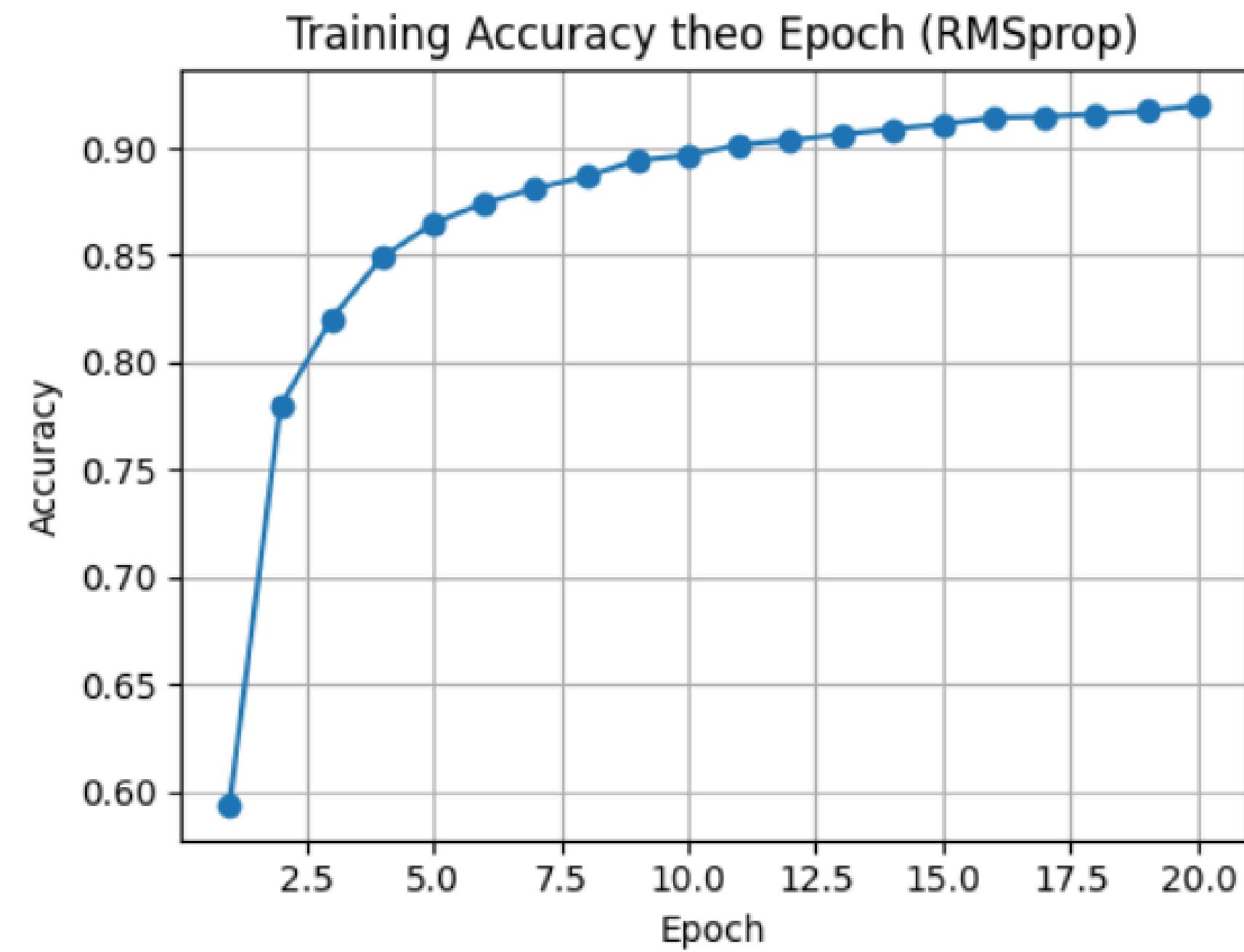
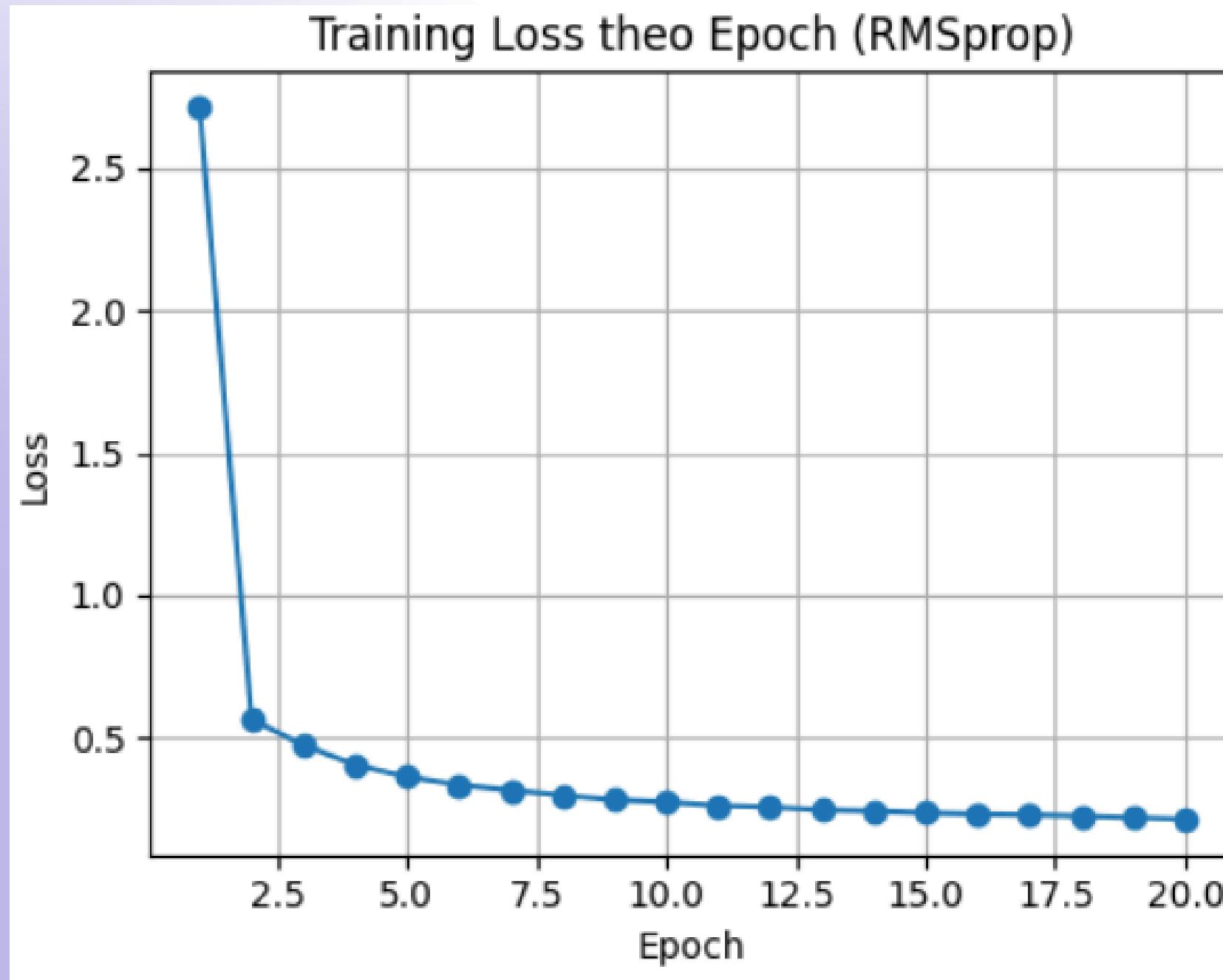
Mở rộng định nghĩa \mathbf{s}_t ta được:

$$\begin{aligned}\mathbf{s}_t &= (1 - \gamma) \mathbf{g}_t^2 + \gamma \mathbf{s}_{t-1} \\ &= (1 - \gamma) (\mathbf{g}_t^2 + \gamma \mathbf{g}_{t-1}^2 + \gamma^2 \mathbf{g}_{t-2}^2 + \dots).\end{aligned}$$

- RMSProp không tích lũy toàn bộ gradient như Adagrad
 - Mỗi gradient quá khứ được giảm trọng số theo γ^k
- Giúp RMSProp tránh phân rã tốc độ học quá mạnh như Adagrad

BÀI TẬP

Training với RMSProp gọi từ Torch



CẢM ƠN VÌ
ĐÃ LẮNG NGHE!

TÀI LIỆU THAM KHẢO

- [1] Aston Zhang, Zachary C. Lipton, Mu Li, và Alexander J. Smola. Dive into Deep Learning. Cambridge University Press, 2023. Truy cập tại: <https://D2L.ai>
- [2] John Duchi, Elad Hazan, và Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, vol. 12, 2011, pp. 2121-2159. Truy cập tại: <https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>
- [3] Valentino, Nico và Setiawan, Erwin. (2024). Movie Recommender System on Twitter Using Weighted Hybrid Filtering and GRU. Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control. 159-172. 10.22219/kinetik.v9i2.1941. Truy cập tại: https://www.researchgate.net/publication/385734477_Movie_Recommender_System_on_Twitter_Using_Weighted_Hybrid_Filtering_and_GRU

TÀI LIỆU THAM KHẢO

- [4] M. A. Mochinski, J. Paul Barddal and F. Enembreck. Improving Multiple Time Series Forecasting with Data Stream Mining Algorithms. 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 2020, pp. 1060-1067. Truy cập tại:
<https://ieeexplore.ieee.org/document/9283059>
- [5] Hongwei Yong. Learning Rate Re-scheduling for AdaGrad in Training Deep Neural Networks. OpenReview, 16 Sept. 2023 (modified 11 Mar. 2024), ICLR 2024. Truy cập tại
<https://openreview.net/forum?id=Nh6pXEkZkK>