NGHIÊN CỨU SO SÁNH CÁC THUẬT TOÁN TỐI ƯU HÓA ADADELTA VÀ ADAM

(Comparative Study of Optimization Algorithms: AdaDelta and Adam)

Nguyễn Tấn Phú[1], Nguyễn Đăng Khoa[1],
Phạm Hoàng Sơn[1], Nguyễn Hoàng Nam[1], Nguyễn Trần Huy Việt[1]

[1]Khoa khoa học và kỹ thuật máy tính

Môn học: Cơ sở toán cho Khoa học máy tính

Nội dung

Nền tảng toán học của bài toán tối ưu trong học máy:

- Huấn luyện mô hình học sâu được mô hình hoá như bài toán tối thiểu hoá hàm mất mát được xác định trên không gian tham số $\mathbb{R}^d$.
- Mục tiêu: tìm tham số θ sao cho giá trị của $f(\theta)$ đạt cực tiểu.
- Các thuật toán dựa trên gradient quyết định tốc độ hội tụ, độ ổn định và hiệu quả tối ưu.

$\Rightarrow$ Tuy nhiên, trong các mô hình phức tạp, những phương pháp cơ sở thường gặp hạn chế về bước học, phương sai gradient và khả năng thích nghi theo từng chiều.

Bài toán tối ưu tổng quát

Cơ sở toán học của quá trình huấn luyện mô hình. Bài toán tối ưu được biểu diễn bởi:

$$\min_{\theta \in \mathbb{R}^d} f(\theta) = \mathbb{E}_{(x,y)\sim D}\Big[\ell(\theta; x, y)\Big].$$

Trong đó:

- θ: vector tham số (thuộc không gian vectơ $\mathbb{R}^d$).
- $f(\theta)$: hàm mất mát cần tối thiểu hoá.
- $\ell(\theta; x, y)$: hàm mất mát tại từng mẫu dữ liệu.

$\Longrightarrow$ Bài toán là tìm θ sao cho gradient tiến về 0

1. Gradient Descent (GD)

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$$

Hạn chế:

- Cần gradient toàn bộ dữ liệu $\longrightarrow$ chi phí lớn.
- Phụ thuộc mạnh vào α:
  - α quá lớn $\longrightarrow$ dãy θ_t không hội tụ.
  - α quá nhỏ $\longrightarrow$ hội tụ chậm.

2. Stochastic Gradient Descent (SGD)

$$g_t = \nabla_\theta \ell(\theta_t; x_t, y_t), \qquad \theta_{t+1} = \theta_t - \alpha g_t.$$

Hạn chế:

- Gradient có phương sai lớn $\longrightarrow$ đường đi dao động.
- Khó tối ưu trong các bài toán có điều kiện kém.

3. Momentum

$$v_t = \beta v_{t-1} + (1-\beta) g_t, \qquad \theta_{t+1} = \theta_t - \alpha v_t.$$

Hạn chế:

- Tốc độ học α vẫn không thích nghi theo từng chiều.
- Có thể vượt quá điểm tối ưu nếu gradient thay đổi đột ngột.

4. AdaGrad

$$G_t = \sum_{i=1}^{t} g_i^2, \qquad \theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} g_t.$$

Hạn chế của AdaGrad:

- G_t tăng không giới hạn $\rightarrow$ bước học giảm dần về 0.
- Không hiệu quả cho tối ưu dài hạn.

$\Rightarrow$ Từ các hạn chế của các thuật toán cơ sở là động lực để phát triển AdaDelta và Adam.

AdaDelta là gì?

- Được đề xuất bởi Matthew D. Zeiler (2012) như một cải tiến của Adagrad.
- Thuộc nhóm *thuật toán tối ưu thích nghi* (adaptive optimization).
- Mục tiêu chính:
  - Khắc phục hiện tượng **learning rate hiệu dụng giảm về 0** của Adagrad.
  - Tự điều chỉnh bước nhảy theo từng chiều mà không cần chọn trước η.

Các đại lượng chính trong AdaDelta

- ρ: hệ số suy giảm trong EMA (*Exponential Moving Average*), điều chỉnh "độ nhớ" lịch sử gradient (thường $0.9 \sim 0.95$).
- ϵ: số rất nhỏ tránh chia cho 0 và làm mượt mẫu số.
- $E[g_t^2]$: EMA của bình phương gradient – tương tự G_t của Adagrad nhưng **không tăng vô hạn**.
- $E[\Delta x_t^2]$: EMA của bình phương bước cập nhật – dùng để chuẩn hoá bước nhảy.
- g_t: gradient tại bước t; Δx_t: bước cập nhật; x_t: trọng số mô hình.

Bước 1: Ước lượng bình phương gradient (EMA)

$$E[g_t^2] = \rho\, E[g_{t-1}^2] + (1-\rho)\, g_t^2$$

- Thay vì cộng dồn tất cả g_τ^2 như Adagrad, AdaDelta chỉ giữ lại **tỉ lệ** ρ của giá trị cũ.
- Phần $(1-\rho)g_t^2$ bổ sung thông tin mới ở bước hiện tại.
- Kết quả: $E[g_t^2]$ phản ánh *lịch sử gần đây* của gradient nhưng **không tăng vô hạn theo** t.

Bước 2: Tính bước cập nhật thích nghi

$$\Delta x_t = -\frac{\sqrt{E[\Delta x_{t-1}^2] + \epsilon}}{\sqrt{E[g_t^2] + \epsilon}} g_t$$

- Mẫu số $\sqrt{E[g_t^2]}$ điều chỉnh **theo độ lớn gradient** từng chiều (giống Adagrad).
- Tử số $\sqrt{E[\Delta x_{t-1}^2]}$ đưa **thang đo của bước cập nhật quá khứ** vào, tạo ra *learning rate động*.

Bước 2: Tính bước cập nhật thích nghi

- Không còn xuất hiện η tường minh như trong SGD, Momentum, Adagrad.

Bước 3: Cập nhật EMA của bước nhảy & tham số

$$E[\Delta x_t^2] = \rho\, E[\Delta x_{t-1}^2] + (1-\rho)\,(\Delta x_t)^2$$

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

- Cả $E[g_t^2]$ và $E[\Delta x_t^2]$ đều là EMA $\Rightarrow$ có xu hướng **hội tụ về giá trị hữu hạn**.

Bước 3: Cập nhật EMA của bước nhảy & tham số

- Khi quá trình ổn định, tỉ lệ

$$\alpha_t = \frac{\sqrt{E[\Delta x_{t-1}^2]} + \epsilon}{\sqrt{E[g_t^2]} + \epsilon}$$

xấp xỉ **một hằng số** $\neq 0$.

Vì sao AdaDelta khắc phục được hạn chế của Adagrad?

- Adagrad: $G_{t,i} = \sum_{\tau=1}^{t} g_{\tau,i}^2 \implies G_{t,i}$ tăng xấp xỉ tuyến tính theo t.

$$\eta_{t,i}^{\text{AdaGrad}} = \frac{\eta}{\sqrt{G_{t,i}} + \epsilon} \to 0$$

- AdaDelta: dùng EMA

$$E[g_{t,i}^2] = \rho E[g_{t-1,i}^2] + (1-\rho) g_{t,i}^2$$

nên $E[g_{t,i}^2]$ **hội tụ về một hằng số hữu hạn**.

Vì sao AdaDelta khắc phục được hạn chế của Adagrad?

- Learning rate hiệu dụng

$$\eta_{t,i}^{\text{AdaDelta}} \approx \frac{\sqrt{E[\Delta x_{t-1,i}^2]}}{\sqrt{E[g_{t,i}^2]}} \not\to 0$$

$\Rightarrow$ thuật toán không bị "chết học".

Ví dụ minh hoạ: $f(x) = (x-5)^2$

- Bài toán: $f(x) = (x-5)^2, \quad f'(x) = 2(x-5), \quad x_0 = 20.$
- So sánh:
  - **Adagrad**:
    - $\Delta x_1 = -1.0$ $(20 \to 19)$.
    - Sau 10 bước: $\Delta x_{10} \approx -0.27$, $x_{10} \approx 15.33$.
    - Bước cập nhật giảm nhanh do G_t tăng mạnh.
  - **AdaDelta** (với $\rho = 0.9$, $\epsilon = 10^{-6}$):
    - $\Delta x_1 \approx -0.00316$, $x_1 \approx 19.997$.
    - $\Delta x_{10} \approx -0.00351$, $x_{10} \approx 19.97$.
    - Bước nhảy gần như **không suy giảm** theo t.

Kết luận ngắn gọn về AdaDelta

- Thay **tích luỹ gradient** bằng **EMA** $\Rightarrow$ giữ được thông tin gradient gần đây nhưng **không làm learning rate tiến về 0**.
- Bước cập nhật dùng tỉ lệ

$$\frac{\mathrm{RMS}[\Delta x]_{t-1}}{\mathrm{RMS}[g]_t} = \frac{\sqrt{E[\Delta x_{t-1}^2] + \epsilon}}{\sqrt{E[g_t^2] + \epsilon}}$$

  tạo ra learning rate **thích nghi theo từng tọa độ**.
- Không cần chọn η ban đầu $\Rightarrow$ "*gần như* learning rate free", giảm gánh nặng tuning.

Kết luận ngắn gọn về AdaDelta

- Phù hợp cho các mô hình cần nhiều epoch, tránh hiện tượng "đứng hình" như Adagrad.

Kết quả các bước của Adagrad

t	x_{t-1}	g_t	G_t	$\eta_t = \frac{1}{\sqrt{G_t}}$	Δx_t	x_t
1	20.000	30.000	900.0	0.0333	-1.000	19.000
2	19.000	28.000	1684.0	0.0244	-0.682	18.318
3	18.318	26.636	2393.5	0.0204	-0.544	17.774
4	17.774	25.548	3046.2	0.0181	-0.463	17.311
5	17.311	24.622	3652.4	0.0165	-0.407	16.904
6	16.904	23.808	4219.2	0.0154	-0.366	16.538
7	16.538	23.076	4751.7	0.0145	-0.335	16.203
8	16.203	22.406	5253.7	0.0138	-0.309	15.894
9	15.894	21.788	5728.3	0.0132	-0.288	15.606
10	15.606	21.212	6178.3	0.0127	-0.270	15.336

Kết quả các bước của Adagrad

Nhận xét: G_t tăng rất nhanh khiến η_t giảm mạnh, làm bước cập nhật Δx_t ngày càng nhỏ.

Kết quả các bước của AdaDelta

t	x_{t-1}	g_t	$E[g_t^2]$	$E[\Delta x_{t-1}^2]$	α_t	Δx_t	x_t
1	20.000	30.000	90.000	0.000000	$\approx 1.05\times10^{-4}$	-0.00316	19.997
2	19.997	29.994	171.000	0.000001	$\approx 1.08\times10^{-4}$	-0.00324	19.994
3	19.994	29.987	243.788	0.000002	$\approx 1.10\times10^{-4}$	-0.00330	19.990
4	19.990	29.981	309.293	0.000003	$\approx 1.11\times10^{-4}$	-0.00335	19.987
5	19.987	29.974	368.208	0.000004	$\approx 1.13\times10^{-4}$	-0.00338	19.984
6	19.984	29.967	421.190	0.000005	$\approx 1.14\times10^{-4}$	-0.00341	19.980
7	19.980	29.960	468.834	0.000006	$\approx 1.15\times10^{-4}$	-0.00344	19.977
8	19.977	29.954	511.673	0.000006	$\approx 1.16\times10^{-4}$	-0.00347	19.973
9	19.973	29.947	550.187	0.000007	$\approx 1.16\times10^{-4}$	-0.00348	19.970
10	19.970	29.940	584.808	0.000008	$\approx 1.17\times10^{-4}$	-0.00351	19.966

Kết quả các bước của AdaDelta

Nhận xét: Bước nhảy Δx_t gần như không giảm theo thời gian — learning rate hiệu dụng ổn định.

Thuật toán Adam

Cho chuỗi giá trị $\{g_t\}_{t=1}^{\infty}$, Exponential Moving Average (EMA) được định nghĩa:

$$v_t = \beta v_{t-1} + (1-\beta) g_t, \quad \beta \in [0,1] \tag{1}$$

Trong đó:

- v_t: giá trị EMA tại thời điểm t;
- v_{t-1}: giá trị EMA tại thời điểm $t-1$;
- g_t: giá trị gradient tại thời điểm t;
- β: hệ số làm mịn;
- $1-\beta$: trọng số của giá trị hiện tại.

Thuật toán Adam

$$
\begin{aligned}
v_1 &= \beta v_0 + (1-\beta) g_1 \\
v_2 &= \beta v_1 + (1-\beta) g_2 \\
&\dots \\
v_5 &= \beta v_4 + (1-\beta) g_5 \\
&= \beta \cdot (1-\beta)[\beta^3 g_1 + \beta^2 g_2 + \beta g_3 + g_4] + (1-\beta) g_5 \\
&= (1-\beta) g_5 + \beta(1-\beta) g_4 + \beta^2 (1-\beta) g_3 + \beta^3 (1-\beta) g_2 + \beta^4 (1-\beta) g_1 \\
&= (1-\beta)[g_5 + \beta g_4 + \beta^2 g_3 + \beta^3 g_2 + \beta^4 g_1]
\end{aligned}
$$

Công thức tổng quát: Tổng quát hóa quy luật trên cho bước thời gian t bất kỳ:

$$v_t = (1-\beta) \sum_{i=1}^{t} \beta^{t-i} g_i$$

Thuật toán Adam

- **First moment** (m_t): EMA của gradient → Momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

- **Second moment** (v_t): EMA của bình phương gradient → RMSProp

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

- **Bias Correction** : Loại bỏ bias do khởi tạo từ zero → đặc biệt quan trọng ở các iteration đầu tiên.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

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

Thuật toán Adam

Cơ chế cập nhập của Adam:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Ý nghĩa:

- $\hat{m}_t$: Hướng di chuyển (có momentum)
- $\sqrt{\hat{v}_t}$: Điều chỉnh độ lớn bước nhảy
- α: Bước nhảy
- ϵ: Số rất nhỏ để tránh chia cho 0 (1e-8)
- β_1: Hệ số EMA cho momentum (thường $= 0.9$)
- β_2: Hệ số EMA cho RMSProp (thường $= 0.999$)

Hạn chế của Adam

- Learning rate hiệu dụng tăng quá nhanh do v_t giảm mạnh khi gradient nhỏ.
- Không ổn định trong môi trường gradient biến động, dễ gây dao động bước nhảy.
- Có thể không hội tụ ngay cả trên bài toán lồi đơn giản.
- Nhạy cảm với siêu tham số như β_2 và ε.

Thuật toán Yogi

Cơ chế cập nhật moment bậc hai của Yogi:

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

Ý nghĩa toán học:

- Nếu $v_{t-1} < g_t^2 \rightarrow$ Yogi *tăng* v_t một lượng nhỏ.
- Nếu $v_{t-1} > g_t^2 \rightarrow$ Yogi *giảm* v_t, tránh moment phình to.
- v_t tiến về giá trị cân bằng, phản ánh trung bình thực sự của gradient.

Moment bậc nhất & cập nhật tham số

Moment bậc nhất giữ nguyên như Adam:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Cập nhật tham số:

$$\theta_{t+1} = \theta_t - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Nguyên lý chính của Yogi:

- Không để moment bậc hai tăng không giới hạn.
- Giữ ổn định learning rate hiệu dụng.
- Tránh triệt tiêu bước cập nhật.
- Cải thiện hội tụ khi gradient không ổn định.

Thuật toán Yogi

Algorithm 1: YOGI Optimizer

Input : Initial point $x_1 \in \mathbb{R}^d$, learning rates $\{\eta_t\}_{t=1}^T$, hyperparameters $0 < \beta_1, \beta_2 < 1$, $\epsilon > 0$

Initialize: $m_0 = 0$, $v_0 = 0$

for $t = 1$ **to** T **do**

1. Sample s_t from data distribution P
2. Compute stochastic gradient: $g_t = \nabla \ell(x_t, s_t)$
3. Update first moment estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
4. Update second moment estimate (Yogi update):
$$v_t = v_{t-1} + (1 - \beta_2)\, g_t^2 \odot \operatorname{sgn}(g_t^2 - v_{t-1})$$
5. Update parameters: $x_{t+1} = x_t - \eta_t \dfrac{m_t}{\sqrt{v_t} + \epsilon}$

Mô phỏng sự khác biệt Adam vs Yogi

Mục tiêu là quan sát cách Adam và Yogi cập nhật mô-men bậc hai v_t và tác động của nó lên tốc độ học hiệu dụng $\eta_t = \frac{1}{v_t + \varepsilon}$, đặc biệt trong bối cảnh gradient biến động đột ngột.

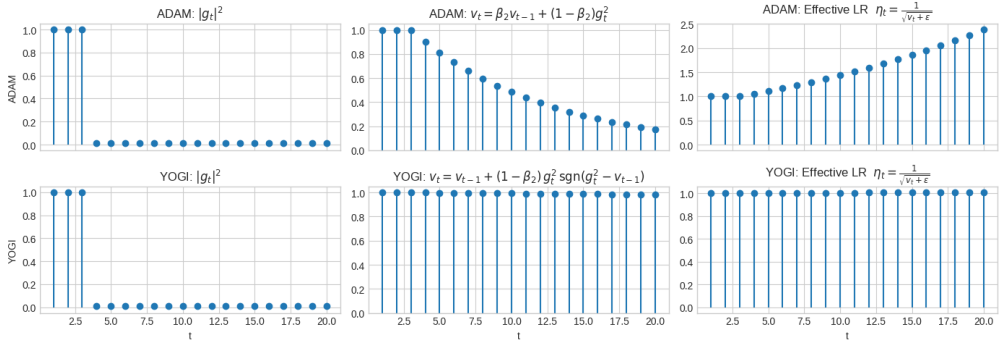
Sử dụng chuỗi gradient tổng hợp:

- Ba bước đầu: $|g_t|^2 = 1.0$ (gradient spike).
- Các bước sau: $|g_t|^2 = 0.01$ (gradient nhỏ ổn định).

Cập nhật mô-men bậc hai được cho bởi:

- Adam: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
- Yogi: $v_t = v_{t-1} + (1 - \beta_2) g_t^2 \odot \operatorname{sgn}\left(g_t^2 - v_{t-1}\right)$

Mô phỏng sự khác biệt Adam vs Yogi



Hình 1: Minh họa sự khác biệt Adam vs Yogi.

Mô phỏng sự khác biệt Adam vs Yogi

Adam: phản ứng mạnh với gradient nhỏ $\longrightarrow$ tốc độ học tăng quá nhanh $\longrightarrow$ dễ gây dao động.

Yogi: kiểm soát quá trình cập nhật v_t $\longrightarrow$ tốc độ học tăng từ tốn $\longrightarrow$ hội tụ ổn định hơn.

$\Longrightarrow$ Trong môi trường có biến động gradient hoặc phân bố không ổn định, **Yogi** thường vượt trội hơn **Adam**.

So sánh và đánh giá thực nghiệm

Thiết lập chung:

- Framework: PyTorch.
- Epochs: 20 vòng lặp cho tất cả các mô hình.
- Batch size: 128.
- Dữ liệu: Giữ nguyên kiến trúc mạng và quy trình tiền xử lý để đảm bảo tính công bằng.

Cấu hình thuật toán tối ưu:

- **Nhóm không thích nghi:** Learning rate $\alpha = 0.01$. Gồm: SGD, SGD + Momentum ($\gamma = 0.9$), SGD + Nesterov.
- **Nhóm thích nghi:** Learning rate $\alpha = 0.001$. Gồm: RMSProp, Adagrad, Adam, Yogi.
- **Ngoại lệ:** Adadelta được thiết lập learning rate = 1.0 theo công bố gốc.

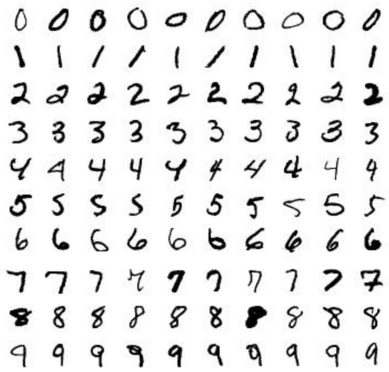
Tập dữ liệu thực nghiệm (MNIST)

Đặc tả dữ liệu:

- **Nội dung:** 70,000 ảnh chữ số viết tay (0-9).
- **Phân chia:** 60,000 ảnh huấn luyện (train) và 10,000 ảnh kiểm tra (test).
- **Kích thước:** 28×28 pixel, đơn kênh (grayscale).

Tiền xử lý:

- Chuẩn hóa giá trị pixel về dải $[0, 1]$.
- Standardize theo thống kê của tập train.
- **Lưu ý:** Không áp dụng Data Augmentation để giữ tính thuần nhất khi so sánh thuật toán.



Hình 2: Minh họa hình ảnh từ bộ dữ liệu thực nghiệm.

Thực nghiệm 1: Multilayer Perceptron (MLP)

Bảng 4.1: Kết quả thực nghiệm của các thuật toán tối ưu trên mô hình MLP

Optimizer	Train Loss	Train Acc	Val Loss	Val Acc	Test Acc
SGD	0.1285	0.9623	0.1490	0.9583	0.9580
SGD + Momentum	0.0010	1.0000	0.1006	0.9784	0.9790
SGD + Nesterov	0.0010	1.0000	0.0997	0.9790	0.9775
RMSprop	0.0132	0.9963	0.1390	0.9771	0.9797
Adagrad	0.1566	0.9541	0.1718	0.9491	0.9514
Adadelta	0.0000	1.0000	0.1330	0.9817	0.9819
Adam	0.0126	0.9961	0.1189	0.9771	0.9794
Yogi	0.0001	1.0000	0.0975	0.9818	0.9804

So sánh và đánh giá thực nghiệm

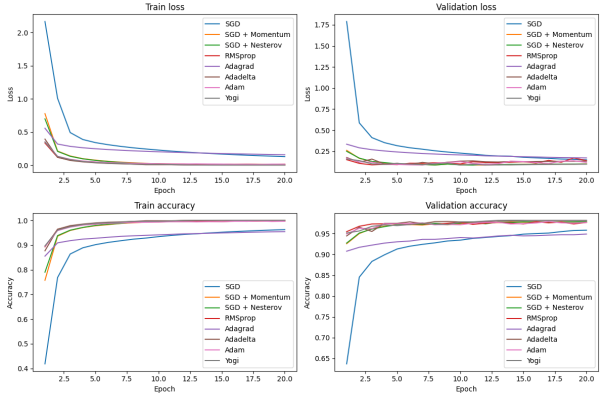
Nhóm cơ sở

- SGD: chậm, Best Acc ≈ 0.958
- Momentum/Nesterov: nhanh, Acc ≈ 0.978–0.979
- RMSprop: nhanh, Acc ≈ 0.979, nhưng dao động
- Adagrad: ổn định, Acc ≈ 0.949

Nhóm thích nghi

- Adadelta: rất nhanh (2–3 epoch), Acc ≈ 0.9818, hơi overfit
- Adam: hội tụ nhanh, Acc ≈ 0.9795
- Yogi: ổn định nhất, Acc ≈ 0.9822

Biểu đồ kết quả: Multilayer Perceptron (MLP)



Kết luận

- Adadelta, Adam, Yogi vượt SGD và Adagrad
- Yogi cao nhất; Adadelta nhanh nhất; Adam cân bằng

Thực nghiệm 2: Convolutional Neural Networks (CNN)

Bảng 4.2: Kết quả thực nghiệm các thuật toán tối ưu với mô hình CNN

Optimizer	train_loss	train_acc	val_loss	val_acc	test_acc
SGD	0.0320	0.9917	0.0527	0.9840	0.9860
SGD + Momentum	0.0052	0.9992	0.0515	0.9869	0.9876
SGD + Nesterov	0.0052	0.9991	0.0579	0.9855	0.9889
RMSprop	0.0056	0.9982	0.0673	0.9855	0.9893
Adagrad	0.0666	0.9829	0.0795	0.9777	0.9810
Adadelta	0.0020	0.9995	0.0694	0.9873	0.9896
Adam	0.0117	0.9958	0.0700	0.9848	0.9878
Yogi	0.0024	0.9999	0.0519	0.9871	0.9895

So sánh và đánh giá thực nghiệm

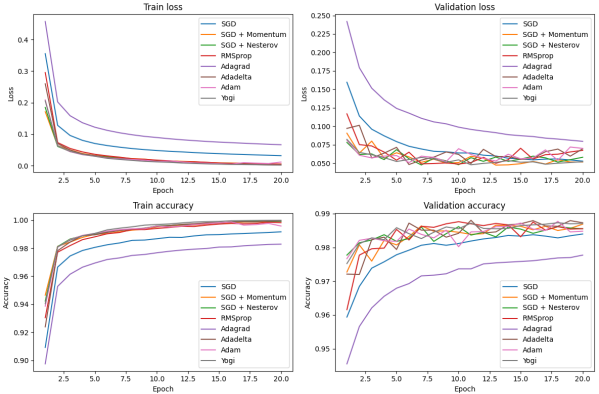
Nhóm cơ sở

- SGD: ổn định, chậm — val 0.9840, test 0.9860
- Momentum / Nesterov: nhanh — 0.9876 / 0.9889
- RMSprop: mạnh — 0.9893; dao động cuối
- Adagrad: yếu nhất — 0.9810

Nhóm thích nghi

- **Adadelta**: tốt nhất — test 0.9896, val 0.9873; ổn định, ít nhạy LR
- **Adam**: nhanh nhất — 0.9878; val_loss dao động
- **Yogi**: ổn định + mạnh — 0.9895, val 0.9871

Biểu đồ kết quả: Convolutional Neural Networks (CNN)



Kết luận

- Adadelta: hiệu năng tổng thể tốt nhất; Adam: tốc độ hội tụ nhanh nhất
- Yogi: cân bằng giữa hiệu năng và ổn định

Thực nghiệm 3: Deep Residual Networks (ResNet-18)

Optimizer	Train Loss	Train Acc	Val Loss	Val Acc	Test Acc
SGD	0.0007	1.0000	0.0530	0.9864	0.9870
SGD + Momentum	0.0002	0.9999	0.0448	0.9896	0.9929
SGD + Nesterov	0.0026	0.9990	0.0424	0.9905	0.9919
RMSprop	0.0094	0.9973	0.0544	0.9889	0.9923
Adagrad	0.0005	1.0000	0.0580	0.9845	0.9846
Adadelta	0.0017	0.9996	0.0411	0.9916	0.9936
Adam	0.0087	0.9971	0.0365	0.9918	0.9916
Yogi	0.0013	0.9996	0.0393	0.9918	0.9928

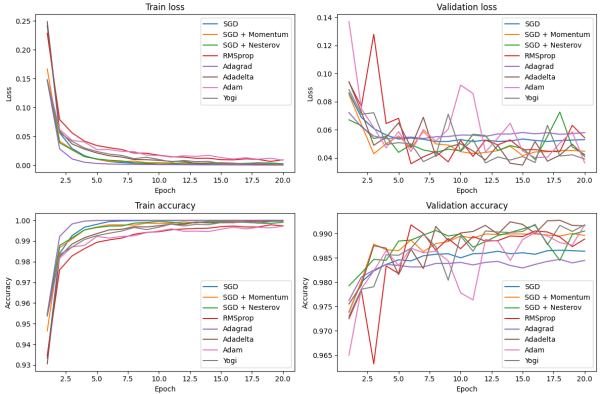
Nhóm cơ sở

- SGD: best val_acc = 0.98656
- SGD + Momentum: best val_acc = 0.99000
- SGD + Nesterov: best val_acc = 0.99167
- RMSprop: best val_acc khoảng 0.992 (hiệu năng cao)
- Adagrad: best val_acc = 0.98469 (thấp nhất)

Nhóm thích nghi

- Adadelta: best val_acc = 0.99271 (cao nhất)
- Adam: best val_acc $\approx$ 0.992 (hội tụ nhanh)
- Yogi: best val_acc $\approx$ 0.992 (ổn định)

Biểu đồ kết quả: Deep Residual Networks (ResNet-18)



Kết luận

- Adadelta, Adam, Yogi: hội tụ nhanh, hiệu năng val cao
- Adadelta tốt nhất; Adam nhanh; Yogi ổn định

Tổng kết đánh giá thực nghiệm

Nhận xét chính từ thực nghiệm:

- Các thuật toán đều hội tụ nhanh, độ chính xác cao
- Momentum và Nesterov cải thiện hội tụ so với SGD
- Nhóm thích nghi huấn luyện nhanh hơn và thích ứng tốt hơn
- Adagrad ổn định nhưng hiệu năng thấp nhất trong nhóm thích nghi

So sánh ba thuật toán:

- Adam: hội tụ nhanh, dễ dùng, nhưng có thể thiếu ổn định
- Adadelta: ít phụ thuộc siêu tham số, bền vững hơn Adam
- Yogi: khắc phục nhược điểm Adam, ổn định tốt trên dữ liệu nhiễu

Gợi ý lựa chọn thuật toán theo bối cảnh ứng dụng

Bài toán đơn giản, gradient ổn định

- Nên dùng SGD, Momentum hoặc Nesterov
- Ưu điểm: ổn định, tổng quát tốt, chi phí thấp

Mô hình sâu hoặc số tham số lớn

- Adam hoặc RMSprop phù hợp
- Ưu điểm: học nhanh, tự điều chỉnh step size

Dữ liệu nhiễu hoặc gradient dao động mạnh

- Yogi ổn định hơn Adam, hạn chế phân kỳ

Bài toán cần giảm phụ thuộc siêu tham số

- Adadelta phù hợp vì không cần chọn learning rate ban đầu

Tài liệu tham khảo I

- L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.
- J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
- T. Tieleman and G. Hinton, "Lecture 6.5 — rmsprop," *COURSERA: Neural Networks for Machine Learning*, 2012. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- M. D. Zeiler, "Adadelta: An adaptive learning rate method," arXiv:1212.5701, 2012.
- D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.
- M. Zaheer, S. J. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," *NeurIPS*, 2018.
- S. Ruder, "An overview of gradient descent optimization algorithms," arXiv:1609.04747, 2016.

Tài liệu tham khảo II

Y. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$," *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *ICML*, pp. 1139–1147, 2013.

J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," *EMNLP*, pp. 1532–1543, 2014.

D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980, 2014.