Making Compute Go Further

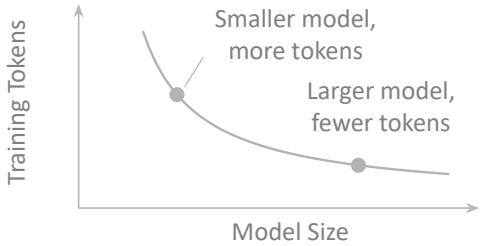
Lecture 3

Anay Mehrotra
MS&E 319
Autumn 2026

Topic 1: Innovations in Pre-Training

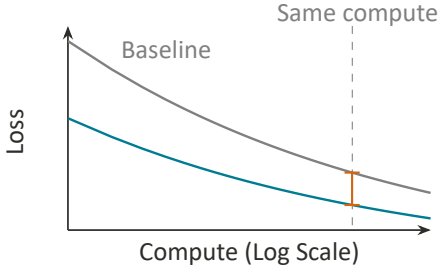
Fix training compute. How do we get the best model from it?

Lecture 2 ✓



How should we split compute between model size and data?

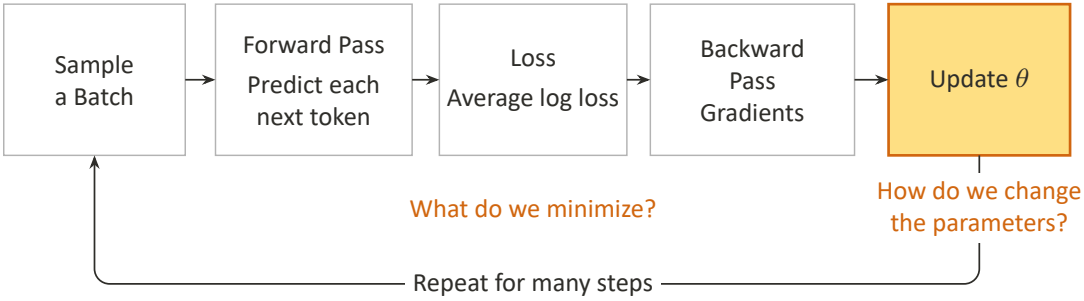
Lecture 3 (Today)



Which optimizer makes compute go further?

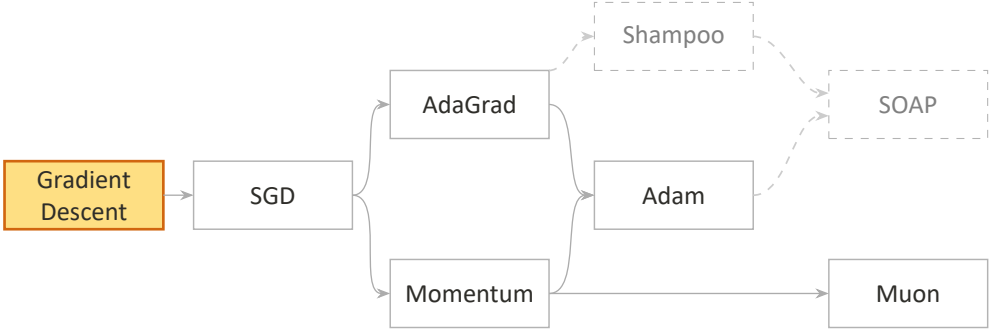
How Training Works

Adjust θ so the model assigns high probability to real text.



In this lecture, we will revisit the parameter update.

Optimizers in Deep Learning



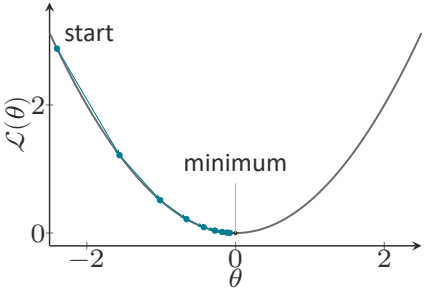
Each optimizer builds on ideas from earlier ones.

First: Gradient Descent.

Gradient Descent

Average loss over n examples:

$$\mathcal{L}(\theta) := \frac{1}{n}\sum_{i=1}^{n} \ell_i(\theta)\,,$$



Gradient descent (GD):

$$\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(\theta_t)\,.$$

step size
(also called learning rate)

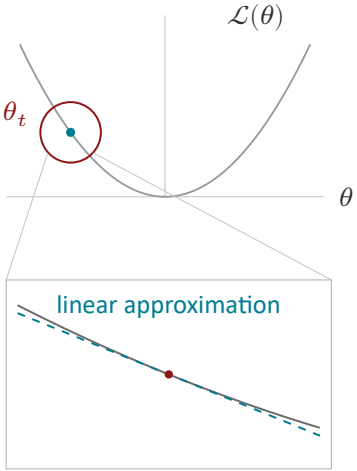
gradient: $\nabla \mathcal{L}(\theta_t) = \left(\frac{\partial \mathcal{L}(\theta_t)}{\partial \theta_{t,1}}, \ldots, \frac{\partial \mathcal{L}(\theta_t)}{\partial \theta_{t,d}} \right)$

Gradient Descent: A Local Method

Gradient descent only looks at the loss near the current point θ_t.

Nearby, the loss is almost linear:

$$\mathcal{L}(\theta_t + \delta) \approx \mathcal{L}(\theta_t) + \nabla \mathcal{L}(\theta_t)^\mathsf{T} \delta \,.$$



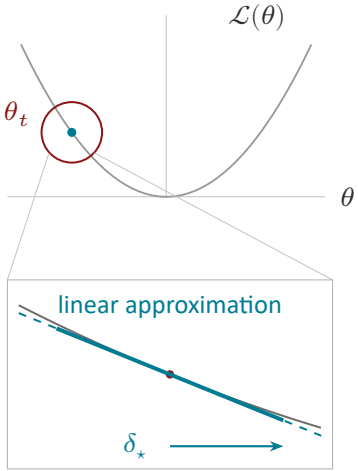
Gradient Descent: A Local Method

Which local step gives the largest reduction?

$$\min_{\|\delta\|_2 \le \rho} \mathcal{L}(\theta_t + \delta)$$
$$\approx \mathcal{L}(\theta_t) + \min_{\|\delta\|_2 \le \rho} \nabla \mathcal{L}(\theta_t)^\mathsf{T} \delta$$

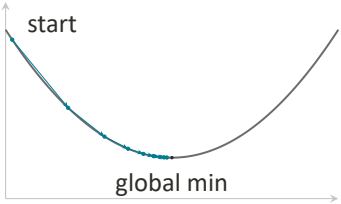
$$\implies \quad \delta_\star = -\rho \frac{\nabla \mathcal{L}(\theta_t)}{\|\nabla \mathcal{L}(\theta_t)\|_2}$$

This is the gradient descent step.



Does GD Find Global Minima?

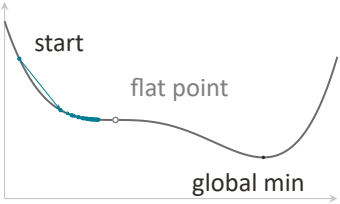
Convex



Yes.

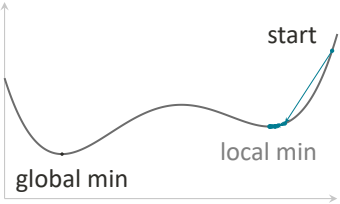
Quasi-convex

(decreases, then increases)



Not necessarily: GD can stop at a nonoptimal flat point.

Nonconvex

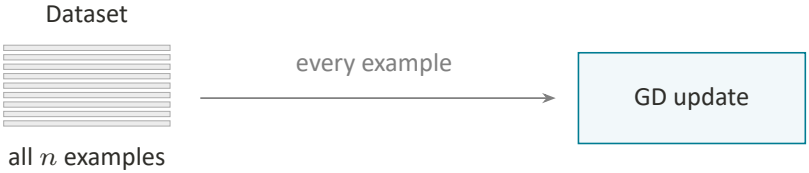


Generally, no.

Neural networks' loss functions are highly nonconvex.

Gradient Descent Is Slow

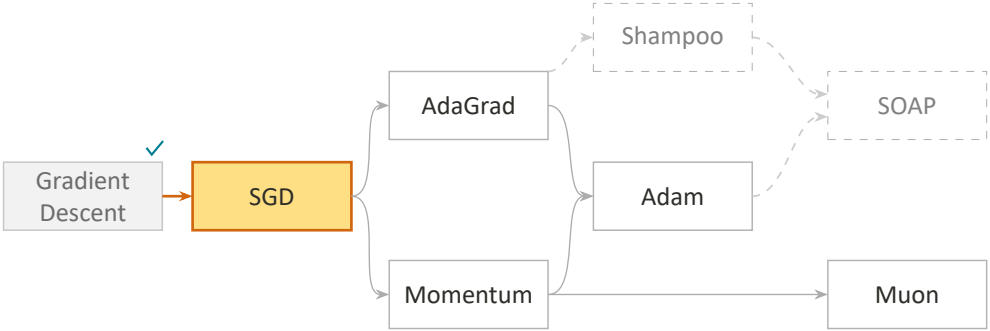
$$\nabla \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell_i(\theta)$$



Training datasets have trillions of tokens.

Can we make gradient descent faster?

Optimizers in Deep Learning



Can we update without reading every example?

Stochastic Gradient Descent

Robbins and Monro (1951)

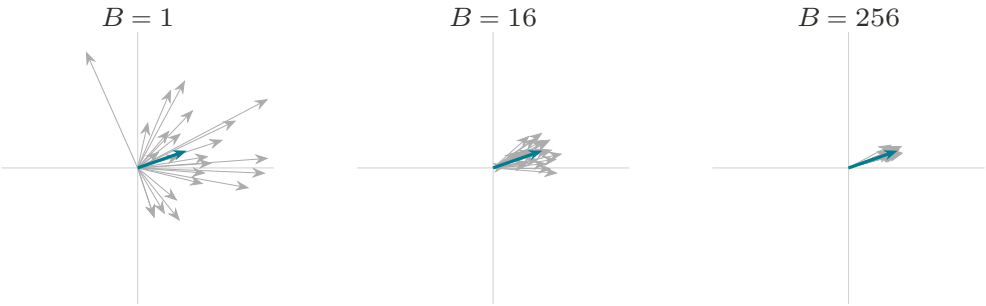
Stochastic gradient descent (SGD): sample a batch $\mathcal{B}_t$ of B examples and average their gradients.

$$g_t := \frac{1}{B} \sum_{i \in \mathcal{B}_t} \nabla \ell_i(\theta_t)\,, \qquad \theta_{t+1} = \theta_t - \eta_t g_t\,.$$

$$\mathbb{E}[g_t] = \nabla \mathcal{L}(\theta_t)\,, \qquad \mathrm{Var}[g_t] \propto 1/B\,.$$

SGD: Averaging Reduces Variance

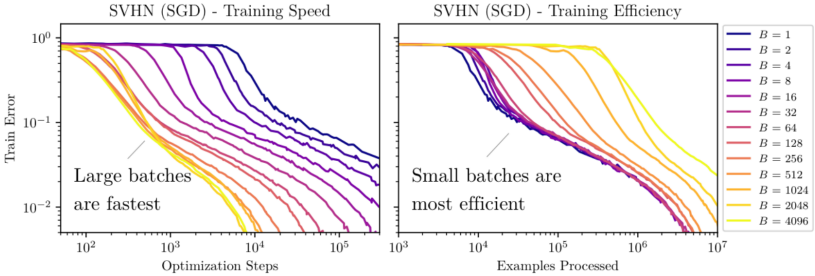
Averaging B independent gradients divides the variance by B.



SGD: How Large Should a Batch Be?

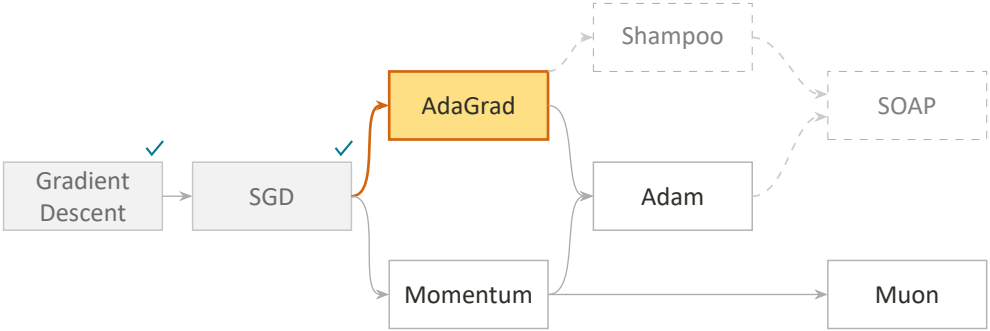
Larger batches need fewer updates but process more examples.

Gains from larger batches plateau at some point.



Source: McCandlish et al. (2018), Fig. 6. CNN on SVHN.

Optimizers in Deep Learning



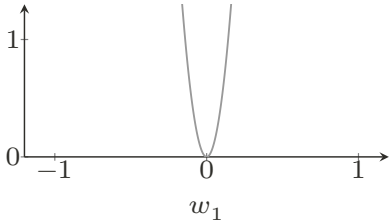
Do different coordinates need different step sizes?

Example: Unequal Curvatures

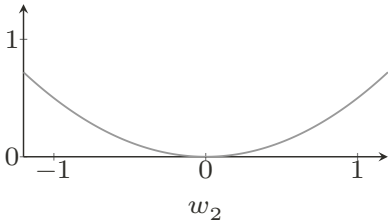
$$f(w) = \tfrac{1}{2}(100w_1^2 + w_2^2)\,, \qquad \nabla f(w) = (100w_1,\, w_2)\,.$$

The loss is a sum of two one-dimensional problems.

w_1: curvature 100



w_2: curvature 1



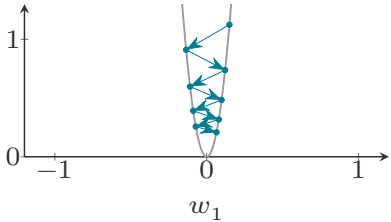
Same Step Size for All Coordinates

Gradient descent uses one step size η for both:

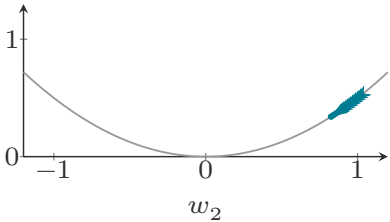
$$w_1 \leftarrow (1 - 100\eta)\, w_1 \,, \qquad w_2 \leftarrow (1 - \eta)\, w_2 \,.$$

Small step size η: stable, but w_2 barely moves.

w_1: curvature 100



w_2: curvature 1

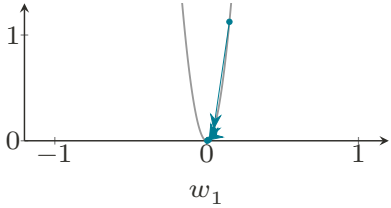


Step Size *Adapts* per Coordinate

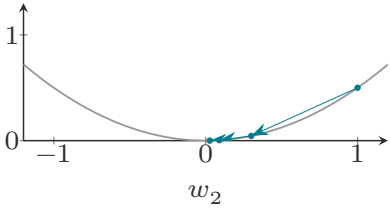
Idea: each coordinate gets its own step size, proportional to $1/\text{curvature}$:

$$w \leftarrow w - \eta \, \mathrm{diag}(1/100,\, 1) \, \nabla f(w) \, .$$

w_1: curvature 100

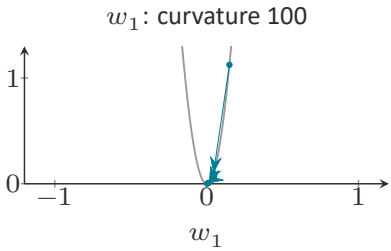


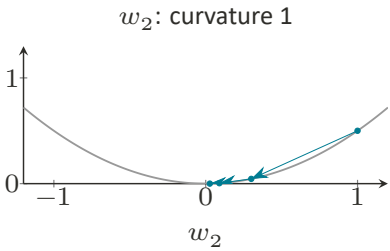
w_2: curvature 1



Step Size *Adapts* per Coordinate

Both coordinates shrink at the same rate.



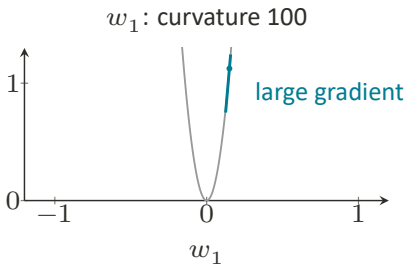


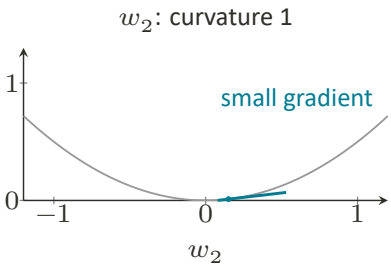
But curvatures are costly to compute.

Using Gradients to Adapt Step Sizes

At the same distance from the minimum, the steeper coordinate has the larger gradient.

Idea: divide each coordinate's step by the size of its gradients.





AdaGrad

Duchi, Hazan, and Singer (2011)

For each coordinate j:

sum of past squared gradients, $s_{t,j} = \sum_{\tau \le t} g_{\tau,j}^2$

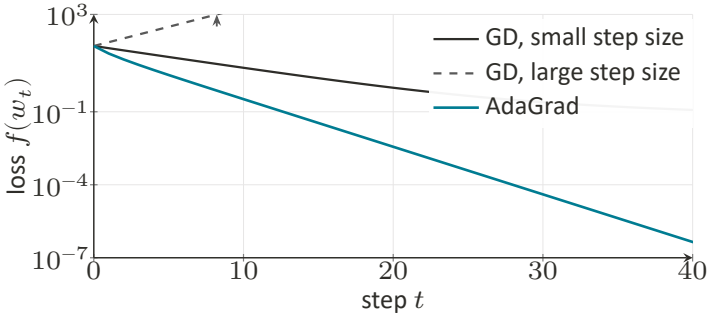
$$s_{t,j} = s_{t-1,j} + g_{t,j}^2 \,,$$

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

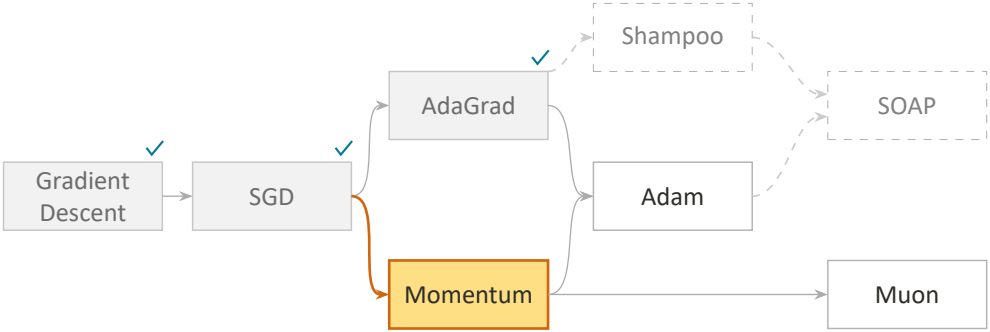
large gradients $\Rightarrow$ small step size

AdaGrad on the Same Example

The curvature cancels: both coordinates shrink at the same rate.



Optimizers in Deep Learning



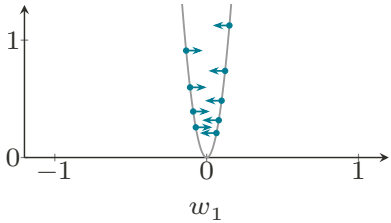
Can averaging past gradients help?

Gradient Descent Zigzags

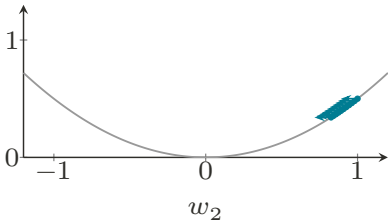
On the steep coordinate, the gradient flips sign every step.

On the shallow coordinate, it keeps its sign.

w_1: curvature 100



w_2: curvature 1



Idea: average recent gradients.

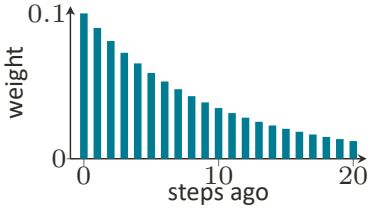
Momentum

Polyak (1964)

Average gradients over time:

$$m_t = \mu m_{t-1} + (1-\mu) g_t \,, \qquad m_0 = 0 \,,$$

$$\theta_{t+1} = \theta_t - \eta_t m_t \,.$$

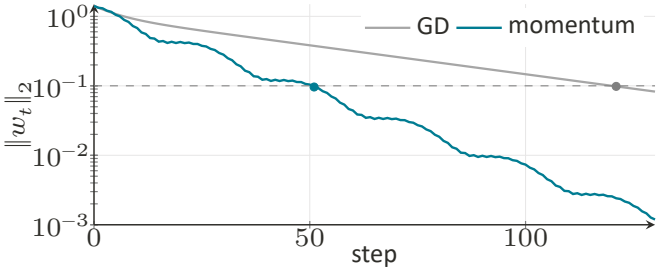


Recent gradients weigh more.

Intuition: consistent gradients “accumulate”; others “cancel”.

Momentum on the Same Example

With momentum, the “steep coordinate” remains stable with a larger step size.



Source: Goh (2017), Why Momentum Really Works.

Momentum Reduces Variance

Suppose each gradient is a fixed mean plus a random fluctuation:

$$g_t = \overline{g} + \xi_t$$

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

$\overline{g}$: fixed (the parameters barely move).

ξ_t: independent across steps and coordinates, mean zero, variance σ^2.

Claim: $\mathrm{Var}(m_t) \to \dfrac{1-\mu}{1+\mu}\sigma^2 < \sigma^2$ (per coordinate, as t grows)

Momentum Reduces Variance

Claim: $\mathrm{Var}(m_t) \to \frac{1-\mu}{1+\mu}\sigma^2$.

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

Proof: Let $V_t = \mathrm{Var}(m_t)$. Since ξ_t is independent of m_{t-1},

$$V_t = \mu^2 V_{t-1} + (1-\mu)^2\sigma^2\,, \qquad V_0 = 0\,.$$

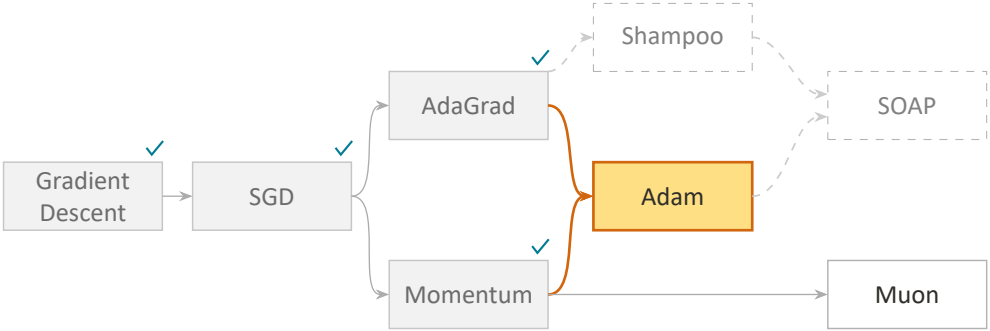
$$V_1 = (1-\mu)^2\sigma^2\,,$$
$$V_2 = (1-\mu)^2\sigma^2\,(1+\mu^2)\,,$$
$$V_3 = (1-\mu)^2\sigma^2\,(1+\mu^2+\mu^4)\,,$$
$$V_t = (1-\mu)^2\sigma^2\,(1+\mu^2+\cdots+\mu^{2(t-1)})\,.$$

Geometric series:

$$1+\mu^2+\mu^4+\cdots = \frac{1}{1-\mu^2}$$

$$V_t \to \frac{(1-\mu)^2}{1-\mu^2}\sigma^2 = \frac{1-\mu}{1+\mu}\sigma^2\,.$$

Optimizers in Deep Learning



Can we combine both ideas?

Adam

Kingma and Ba (2015)

Adam: Adaptive Moment Estimation.

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

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^{\odot 2}$$ ← $\odot 2$: square each coordinate

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \widehat{v}_t = \frac{v_t}{1-\beta_2^t}$$ ← bias correction: $m_0 = v_0 = 0$ make early averages too small

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

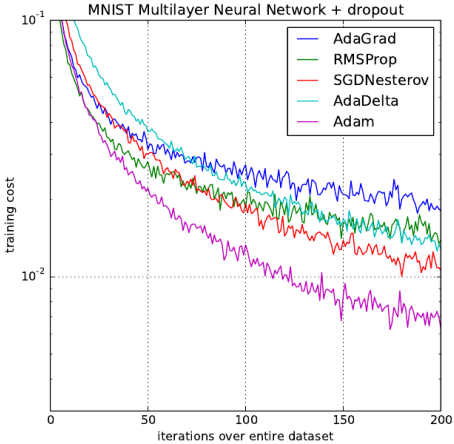
← momentum

↑ AdaGrad's scale, averaged instead of summed

Does Adam Help in Practice?

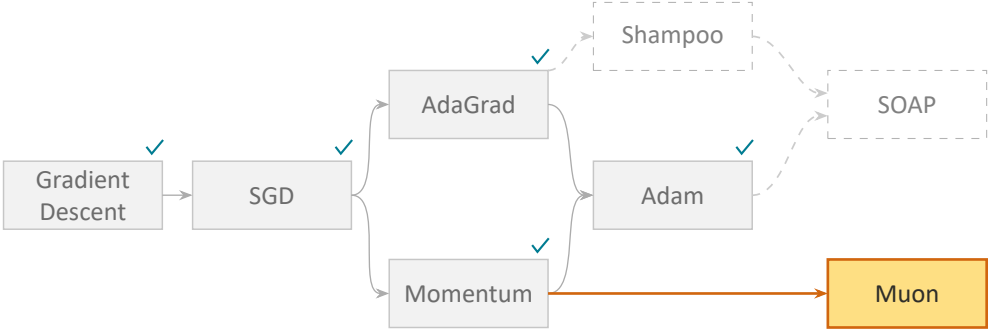
On MNIST networks, Adam trained faster than AdaGrad and SGD with Nesterov momentum.

In practice, most LLMs use AdamW: Adam with weight decay.



Source: Kingma and Ba (2015), Fig. 2. Multilayer networks on MNIST.

Optimizers in Deep Learning



Can we use the matrix structure of each layer?

Gradient Descent: A Local Method

Gradient descent takes the best step within a small ball around θ_t:

$$g = \nabla \mathcal{L}(\theta_t)$$

$$\min_{\|\delta\|_2 \le \rho} \mathcal{L}(\theta_t + \delta) \approx \mathcal{L}(\theta_t) + \boxed{\min_{\|\delta\|_2 \le \rho} g^\mathsf{T} \delta} \longleftarrow \text{optimization viewpoint}$$

$$\implies \quad \delta_\star = -\rho \frac{g}{\|g\|_2}$$

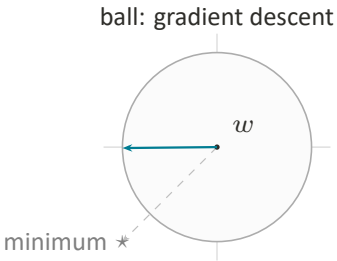
Is the ball $\|\delta\|_2 \le \rho$ the right notion of distance?

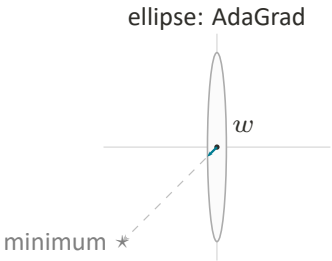
Optimization Viewpoint: AdaGrad

Consider an ellipse constraint:

$$\min_{\sum_j d_j \delta_j^2 \le \rho^2} g^\mathsf{T}\delta \quad \Longrightarrow \quad \delta_{\star,j} \propto -\frac{g_j}{d_j}$$

With $d_j = \sqrt{s_{t,j}}$, this is AdaGrad's step.

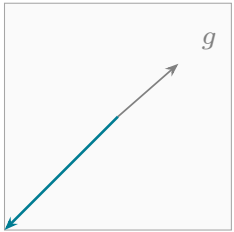




Optimization Viewpoint: Adam

A box: each coordinate may move at most ρ.

$$\min_{\|\delta\|_\infty \le \rho} g^\mathsf{T}\delta \quad \Longrightarrow \quad \delta_\star = -\rho\ \mathrm{sign}(g)$$



Adam without averaging ($\beta_1 = \beta_2 = 0$) takes exactly this step:

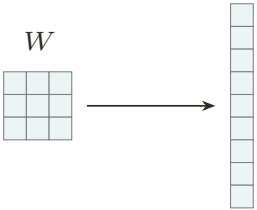
$$\frac{\widehat{m}_t}{\sqrt{\widehat{v}_t}} = \frac{g}{|g|} = \mathrm{sign}(g)$$

Source: Bernstein and Newhouse (2024).

Parameters Are Often Matrices

In deep learning, we often optimize matrices, not vectors.

We could flatten a matrix W into the vector of its entries:



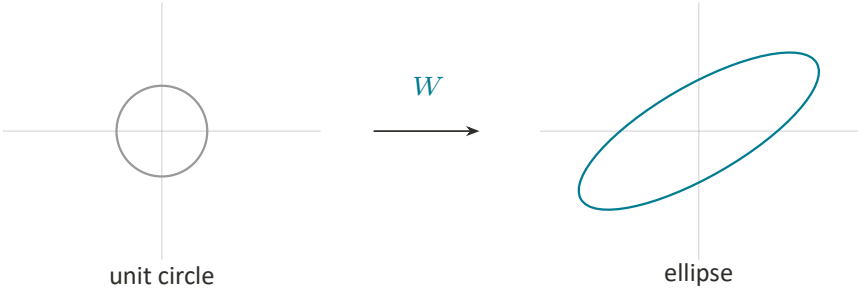
Is this the right way?

Flattening loses the *structure* of the matrix.

Which Distance for Matrices?

A linear layer computes $y = Wx$.

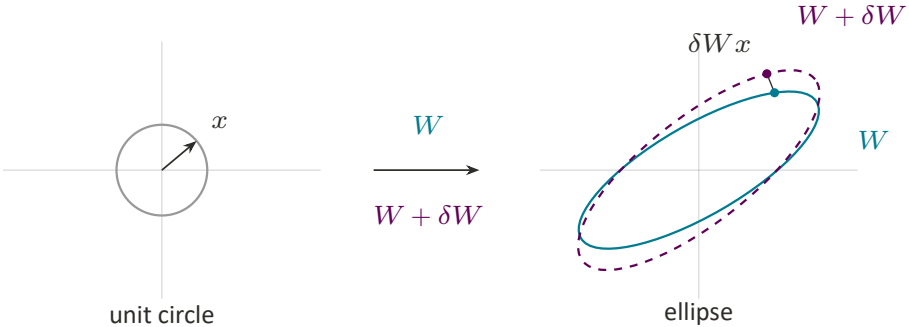
It can rotate, stretch, or collapse directions.



W transforms the unit circle into an ellipse.

Which Distance for Matrices?

How should we measure the distance between W and $W + \delta W$?

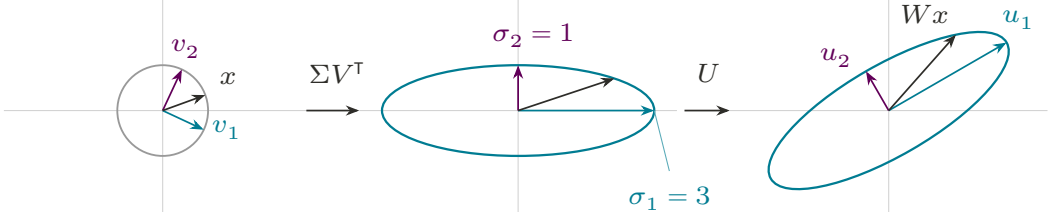


Largest change in output: $\max_{\|x\|_2=1} \|\delta W x\|_2$.

Singular Value Decomposition

$$W = U\Sigma V^\mathsf{T} = \sum_i \sigma_i \, u_i v_i^\mathsf{T}$$

U, V: orthonormal directions (rotations). Σ: diagonal, $\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$.



1. write x in the v_i directions
2. stretch the v_i part by σ_i
3. send it along the output directions u_i

Each input direction v_i is stretched by σ_i and sent to u_i: $Wv_i = \sigma_i u_i$.

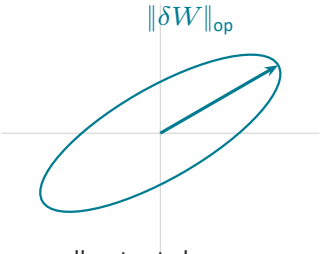
Spectral Norm: The Largest Stretch

$$\|A\|_{\text{op}} := \max_{\|x\|_2=1} \|Ax\|_2 = \sigma_{\max}(A)$$

Proof. Write $x = \sum_i c_i v_i$ with $\sum_i c_i^2 = 1$.

Upper bound: $\|Ax\|_2^2 = \sum_i \sigma_i^2 c_i^2 \leq \sigma_{\max}^2$.

Lower bound: $x = v_1$ gives $\|Av_1\|_2 = \sigma_{\max}$.



all output changes

Optimization Viewpoint: Muon

Which local step gives the largest reduction?

$$\min_{\|\delta W\|_{\text{op}} \le \rho} \mathcal{L}(W + \delta W) \approx \mathcal{L}(W) + \min_{\|\delta W\|_{\text{op}} \le \rho} \langle G, \delta W \rangle$$

$$\langle G, \delta W \rangle = \sum_{i,j} G_{ij}\, \delta W_{ij}$$

Write the gradient's SVD: $G = U \Sigma V^{\mathsf{T}} = \sum_{i=1}^{r} \sigma_i u_i v_i^{\mathsf{T}}$.

Claim: $\delta W_\star = -\rho U V^{\mathsf{T}}$ solves $\min_{\|\delta W\|_{\text{op}} \le \rho} \langle G, \delta W \rangle$.

Optimization Viewpoint: Muon

Claim: $\delta W_\star = -\rho UV^\mathsf{T}$ solves $\min_{\|\delta W\|_{\mathsf{op}} \le \rho} \langle G, \delta W \rangle$.

Proof: Each u_i, v_i is a unit vector, so

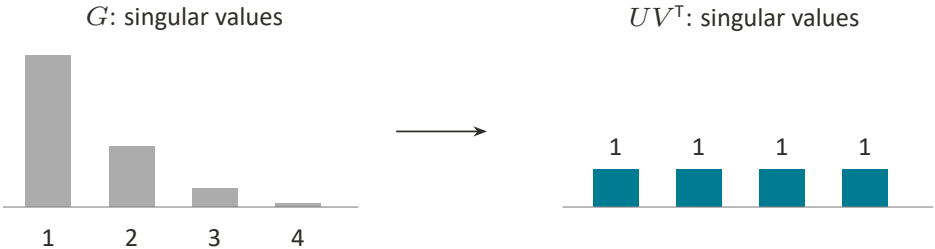
$$u_i^\mathsf{T} \delta W v_i \ge -\|\delta W\|_{\mathsf{op}} \ge -\rho\,.$$

$$\langle G, \delta W \rangle = \sum_{i=1}^{r} \sigma_i u_i^\mathsf{T} \delta W v_i \ge -\rho \sum_{i=1}^{r} \sigma_i\,.$$

$\delta W = -\rho UV^\mathsf{T}$ has spectral norm ρ and gives $u_i^\mathsf{T} \delta W v_i = -\rho$ for every i, so it attains the bound.

What the Best Update Does

$G = U\Sigma V^\top \longmapsto UV^\top \longleftarrow$ the polar factor: $\mathrm{polar}(G) = UV^\top$ (used by Muon)



Keep the singular directions. Set every singular value to 1.

AdaGrad rescaled each coordinate;
Muon rescales each singular direction.

Muon

Jordan et al. (2024)

Muon: MomentUm Orthogonalized by Newton–Schulz.

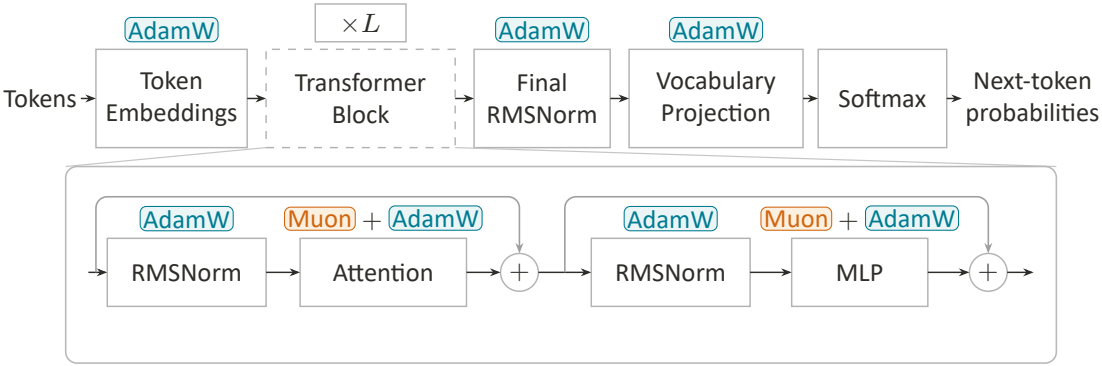
$$M_t = \mu M_{t-1} + (1-\mu) G_t \,,$$

$$W_{t+1} = W_t - \eta_t \, s \, \mathrm{polar}(M_t) \,.$$

sets the update's size

$\mathrm{polar}(U \Sigma V^\mathsf{T}) = U V^\mathsf{T}$: every singular value set to 1

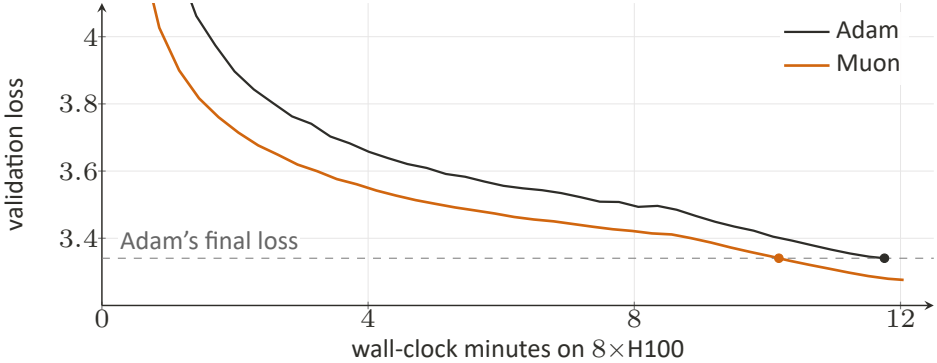
Which Parameters Use Muon?



Muon for matrices; AdamW for everything else.

NanoGPT Experiment

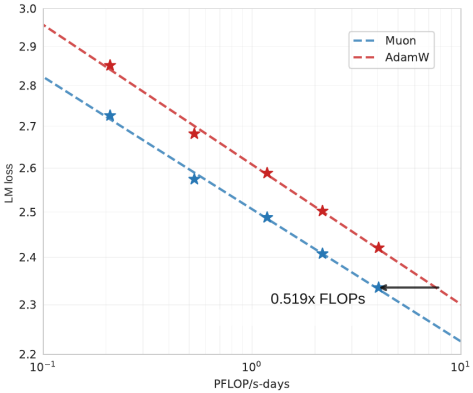
A 124M GPT on 2.7B FineWeb tokens.



Source: Jordan (2024), NanoGPT optimizer comparison.

Muon: Does It Help at Scale?

Moonlight (2025): loss against training FLOPs for 399M–1.5B models.



(a)

Frontier LLMs trained with Muon

- Kimi K2 (2025)
- GLM-4.5 (2025)
- DeepSeek-V4 (2026)
- Kimi K3 (2026)

Source: Liu et al. (2025), Fig. 1(a); Kimi Team (2025); GLM-4.5 Team (2025); DeepSeek-AI (2026); Kimi Team (2026).

Muon in Practice and Research

Computing polar**:** an SVD is slow; a few Newton–Schulz steps approximate it well.

Recent advance: Polar Express (ICLR 2026) converges faster than Newton–Schulz by picking the best polynomial at each iteration.

MuonClip: clips attention logits by rescaling W_Q and W_K (Kimi K2, 2025).

Other matrix-based methods: Sinkhorn balancing (DeepSeek-V4.1 Flash), Shampoo, SOAP.

Source: Jordan et al. (2024); Amsel et al. (2026); Kimi Team (2025); DeepSeek-AI (2026), DeepSeek-V4.1-Flash, §2.5.

Summary

Many optimizers: SGD, AdaGrad, momentum, Adam, AdamW, Muon, …

Recurring ideas:

- adaptive step sizes and momentum
- viewing each update through an *optimization lens*

Different parameter *geometries* need different optimizers.

2012	2017	2023	2025	2026
AlexNet SGD with momentum	Transformer Adam	Llama 2 AdamW	Kimi K2 Muon	DeepSeek-V4.1 Flash Muon, Sinkhorn, AdamW

Questions?

The NanoGPT Speedrun

Time to reach 3.28 validation loss on FineWeb with 8 H100s

Date	Time	Main change
May 2024	45 min	GPT-2 Small baseline (llm.c)
Oct 2024	24.9 min	Muon for hidden-layer matrices
Oct 2024	12.0 min	ReLU^2, QK-norm, faster Muon
Nov 2024	5.0 min	FlexAttention
Dec 2025	1.9 min	FP8, Polar Express
Aug 2026	68 s	Fused kernels, more FP8
Sep 2026	**40 s***	New optimizer, n-gram tables

today's lecture

Lecture 5: Efficient Attention

Lecture 7: Quantization

Lecture 4: Reducing Active Parameters

*Claimed September 2026 (pull request #360). Source: Jordan et al., modded-nanogpt record history.