Deep Learning

Descida de Gradiente

Lucas Silveira Kupssinskü

Machine Learning Theory and Applications Laboratory
PUCRS

agosto de 2026

Visão Geral

Visão Geral

Um exemplo concreto: regressão logística com 1 peso e 1 bias

- Uma única unidade sigmoide com 2 parâmetros (peso w e bias b).
- Função de custo: erro quadrático médio.

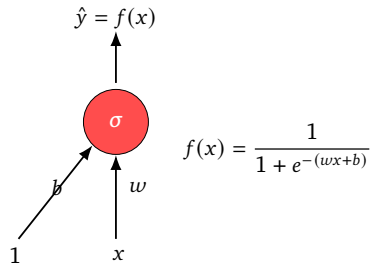
$$\mathcal{L}(w, b) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2$$

- Pergunta natural: por que não simplesmente

$$\underset{w,b}{\operatorname{argmin}} \ \mathcal{L}(w, b)$$

via varredura por força bruta?

- *Vamos tentar* (demo interativa da disciplina)



E se tentássemos varrer todos os (w, b)?

- Com 2 parâmetros conseguimos visualizar a paisagem da loss em uma grade.
- Para 2 parâmetros, com k valores em cada eixo, avaliamos k^2 pontos.
- Para uma rede com P parâmetros, avaliamos k^P pontos.

Para fixar a ideia

Mesmo um MLP modesto com P=1 000 parâmetros e apenas k=10 valores por eixo exigiria 10^{1000} avaliações. Inviável.

Precisamos de um método guiado

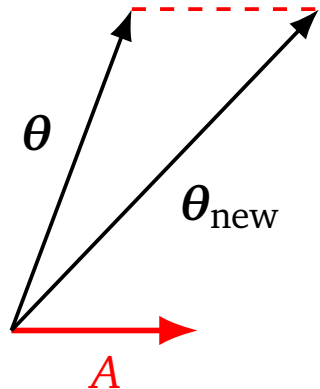
- Força bruta ignora qualquer estrutura da função.
- Em problemas suaves (diferenciáveis), a própria função carrega informação local sobre como reduzir a loss.
- A descida de gradiente é justamente um método que usa essa informação local para escolher a direção e o tamanho do próximo passo.

Visão Geral

Estratégia: aplicar uma atualização que reduza a loss

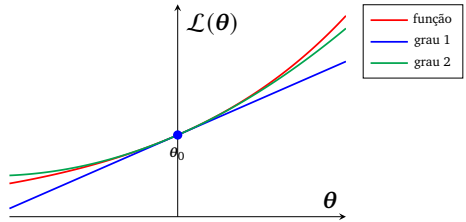
- Queremos achar uma atualização A tal que $\mathcal{L}(\boldsymbol{\theta} + A) < \mathcal{L}(\boldsymbol{\theta})$.
- A pergunta é: qual atualização A escolher?

Aqui $\boldsymbol{\theta} \in \mathbb{R}^n$ é o vetor com todos os n parâmetros do modelo empilhados: cada matriz de pesos e cada vetor de viés entram vetorizados nesse único vetor coluna.



Refresher: Série de Taylor[1]

- Aproxima localmente uma função suave $\mathcal{L}(\boldsymbol{\theta})$ por um polinômio em torno de $\boldsymbol{\theta}_0$.
- Quanto mais termos, melhor a aproximação.
- Para deslocamentos pequenos em torno de $\boldsymbol{\theta}_0$, a primeira ordem domina.



$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla \mathcal{L}(\boldsymbol{\theta}_0) + \frac{1}{2!}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla^2 \mathcal{L}(\boldsymbol{\theta}_0)\,(\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \cdots$$

O termo de primeira ordem é um produto escalar, então a ordem não importa: $(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla \mathcal{L}(\boldsymbol{\theta}_0) = \nabla \mathcal{L}(\boldsymbol{\theta}_0)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$. Muitos livros escrevem com o gradiente transposto.

[1] Demo interativa da disciplina: lucaskup.github.io/deep-learning-course/gradient-descent.

Aplicando Taylor à Loss

Na série de Taylor, expandimos em torno de $\boldsymbol{\theta}_0 = \boldsymbol{\theta}$ e avaliamos no ponto atualizado $\boldsymbol{\theta} + A$:

$$\mathcal{L}(\boldsymbol{\theta} + A) = \mathcal{L}(\boldsymbol{\theta}) + (\boldsymbol{\theta} + A - \boldsymbol{\theta})^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) + \frac{1}{2!}(\boldsymbol{\theta} + A - \boldsymbol{\theta})^\top \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})\,(\boldsymbol{\theta} + A - \boldsymbol{\theta}) + \cdots$$

- Em todos os termos, $\boldsymbol{\theta} - \boldsymbol{\theta}$ se anula: o deslocamento é simplesmente A.

$$\mathcal{L}(\boldsymbol{\theta} + A) = \mathcal{L}(\boldsymbol{\theta}) + A^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) + \frac{1}{2!} A^\top \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})\, A + \cdots$$

- O termo quadrático em A só é desprezível se o passo for pequeno.

Aplicando Taylor à Loss

- Ideia: controlar o tamanho do passo com um escalar, atualizando por ηA com $\eta > 0$ pequeno, a *taxa de aprendizado* (*learning rate*):

$$\mathcal{L}(\boldsymbol{\theta} + \eta A) = \mathcal{L}(\boldsymbol{\theta}) + \eta\, A^{\top} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) + \frac{\eta^2}{2!} A^{\top} \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})\, A + \cdots$$

- Para η suficientemente pequeno, $\eta^2, \eta^3, \ldots$ são desprezíveis e ficamos com a aproximação de primeira ordem:

$$\mathcal{L}(\boldsymbol{\theta} + \eta A) \approx \mathcal{L}(\boldsymbol{\theta}) + \eta\, A^{\top} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}).$$

Critério de melhoria

Queremos que a atualização reduza a loss, ou seja

$$\mathcal{L}(\boldsymbol{\theta} + \eta A) - \mathcal{L}(\boldsymbol{\theta}) < 0.$$

Substituindo a aproximação de primeira ordem:

$$\eta\, A^{\top} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) < 0.$$

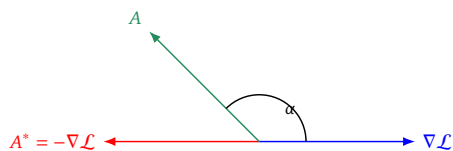
- Como $\eta > 0$, basta exigir $A^{\top} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) < 0$.
- Isto é, o produto escalar entre a direção de atualização e o gradiente deve ser negativo.

Quando o produto escalar é negativo?

Seja α o ângulo entre A e $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$.

$$A^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \|A\| \, \|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})\| \cos(\alpha).$$

- $\|A\| \geq 0$ e $\|\nabla \mathcal{L}\| \geq 0$ sempre.
- Logo, o sinal vem inteiramente de $\cos(\alpha)$.
- Negativo para $\alpha \in (90°, 180°)$.
- "Mais negativo" para $\alpha = 180°$, ou seja $A = -\nabla_{\boldsymbol{\theta}} \mathcal{L}$.



Regra de atualização da Descida de Gradiente

Direção ótima

A escolha que minimiza o produto escalar é $A = -\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$.

Substituindo na atualização $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta\, A$:

$$\boxed{\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)}$$

- Cada passo move os parâmetros na direção oposta ao gradiente.
- η controla o tamanho do passo.
- Esta é a **descida de gradiente**[2].

[2] Ian Goodfellow, Yoshua Bengio e Aaron Courville. Deep Learning. MIT Press, 2016. URL: https://www.deeplearningbook.org/.

Exemplo numérico: descida de gradiente

Cenário

$$\eta = 0.1, \qquad \theta_t = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \qquad \nabla_\theta \mathcal{L}(\theta_t) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

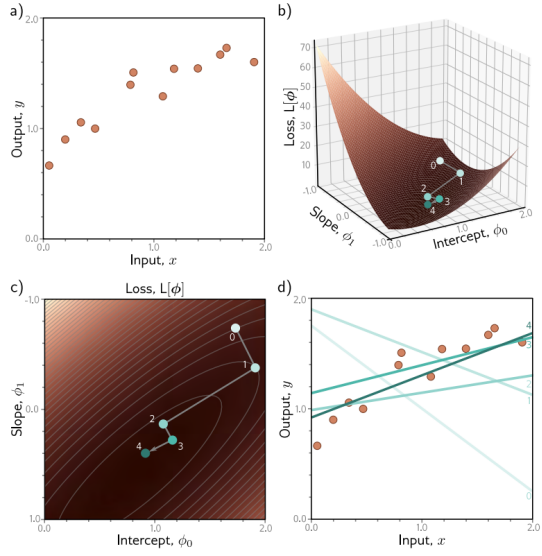
$$\theta_{t+1} = \theta_t - \eta \, \nabla_\theta \mathcal{L}(\theta_t) = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.9 \\ 4.8 \end{bmatrix}.$$

- O passo é proporcional ao gradiente: a segunda coordenada anda o dobro (0.2 contra 0.1).

Visão Geral

Caso 1: função convexa

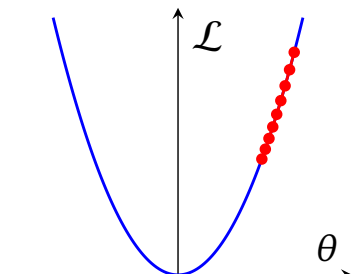
- Modelo linear: $\hat{y} = \theta_0 + \theta_1 x$.
- Loss MSE: $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}(\mathbf{y} - \hat{\mathbf{y}})^\top(\mathbf{y} - \hat{\mathbf{y}})$.
- A superfície tem um único mínimo.
- Qualquer inicialização converge ao mínimo **global**, desde que η seja suficientemente pequeno.



O efeito da taxa de aprendizado

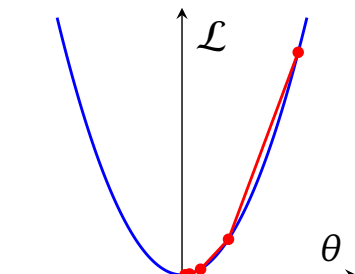
Mesmo no caso convexo, a escolha de η importa. Exemplo com $\mathcal{L}(\theta) = \theta^2$, $\theta_0 = 2$ e atualização $\theta_{t+1} = \theta_t - \eta\, 2\theta_t$:

$\eta = 0.02$



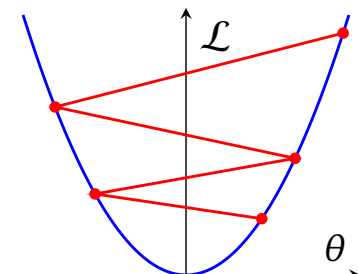
Muito pequena: quase não sai do lugar.

$\eta = 0.3$



Adequada: converge em poucos passos.

$\eta = 1.1$



Muito grande: salta o mínimo e diverge.

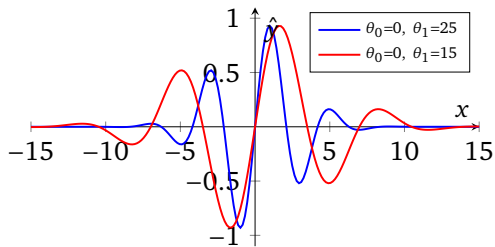
É o mesmo η da dedução via Taylor: a aproximação de primeira ordem só vale para passos pequenos.

Caso 2: função não convexa, modelo de Gabor

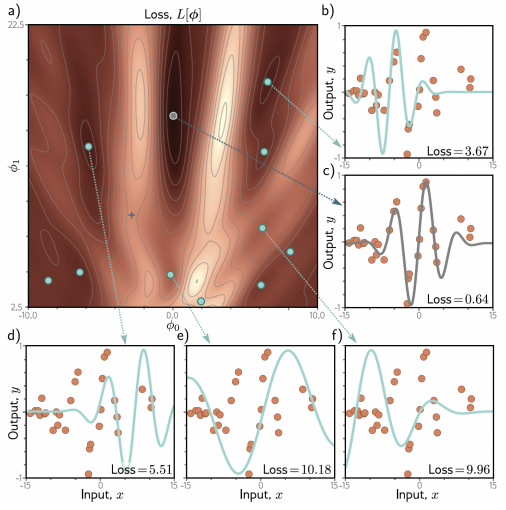
Modelo com apenas 2 parâmetros:

$$\hat{y} = \sin(\theta_0 + 0.06\,\theta_1 x)\ \exp\left(-\frac{(\theta_0 + 0.06\,\theta_1 x)^2}{32}\right)$$

- θ_0 controla o deslocamento.
- θ_1 controla a frequência.
- Apesar de ter só 2 parâmetros, ajustar a curva aos dados já é não trivial.

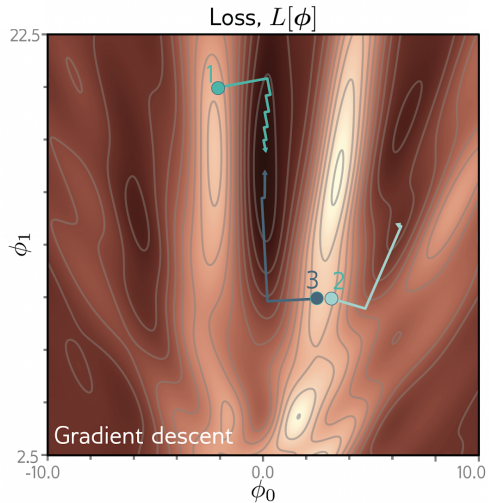


Paisagem da loss do modelo de Gabor[3]



[3] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.

Descida em uma loss não convexa[4]



- Para inicializações em “bons vales”, a descida converge a um mínimo **local** próximo ao global.
- Para inicializações em vales rasos, a descida fica presa.
- *Spoiler*: em redes neurais profundas, a paisagem nunca é convexa.
- Precisamos de variações que ajudem a sair de pontos ruins.

[4] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.

Visão Geral

Variar quantas instâncias entram no gradiente

- Podemos estimar o gradiente com diferentes tamanhos de *batch*:

Tamanho do batch	Nome usual	Atualização
N (todo o dataset)	*Batch Gradient Descent*	determinística
$1 < m < N$	*Mini-batch Gradient Descent*	estocástica
1	*Stochastic Gradient Descent* (SGD)	altamente estocástica

- Em deep learning, "SGD" costuma ser usado como sinônimo de mini-batch com $m \ll N$.

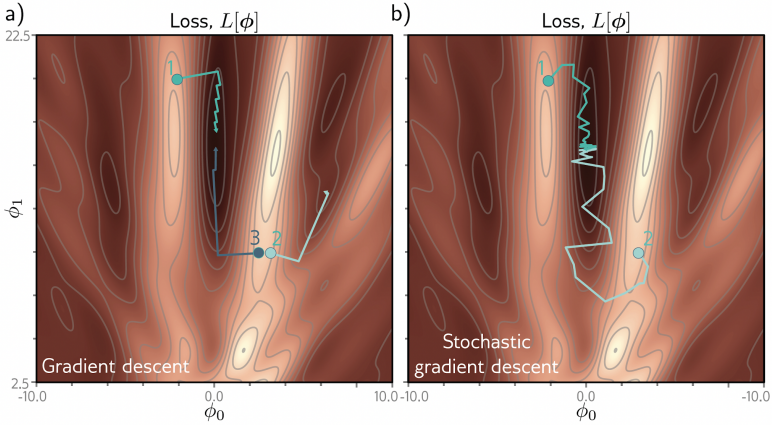
Por que não usar sempre o batch completo?

- Limitações de *hardware*: o batch precisa caber na memória da GPU.
- Custo por passo: cada atualização vê N exemplos, mesmo quando uma fração já seria suficiente para uma boa estimativa do gradiente.
- Falta de ruído: a trajetória é determinística e não consegue escapar de mínimos locais ou pontos de sela.

Trade-off

Mini-batches pequenos introduzem ruído (bom para escapar) e cabem em memória, mas a estimativa do gradiente é mais barulhenta (passos mais erráticos).

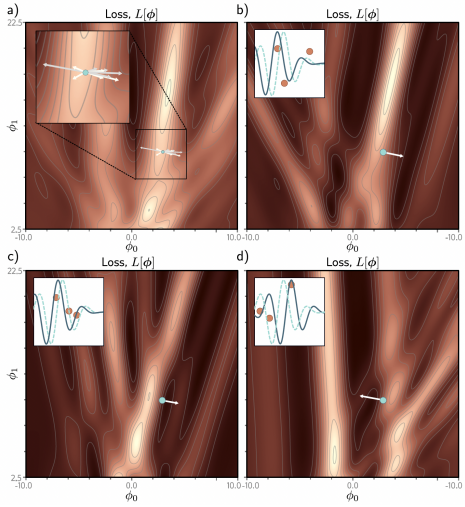
Trajetória: *Batch* vs *Stochastic*[5]



- Esquerda: *batch gradient descent* usa o gradiente exato em cada passo.
- Direita: *stochastic gradient descent* oscila, mas explora vales que o batch ignora.

[5] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.

Mini-batches em uma paisagem real[6]



- Em a) está a loss completa.
- Em b), c), d) vemos diferentes mini-batches com m=3.
- Cada mini-batch aproxima o gradiente verdadeiro com erro, o que se traduz em ruído na trajetória.

[6] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.

Propriedades do SGD

- Instâncias amostradas sem reposição; uma **época** é completada quando todas as instâncias foram vistas.
- Adiciona ruído ao treinamento, ajudando a escapar de mínimos locais e pontos de sela.
- É computacionalmente leve (uma instância por passo).
- Empiricamente, soluções obtidas com SGD generalizam melhor.
- Não converge a um ponto fixo (passos sempre flutuam ao redor do mínimo).

Motivação: usar a memória dos gradientes

- Gradientes consecutivos podem oscilar fortemente (mini-batch ruidoso ou vales estreitos).
- Ideia: somar uma fração da atualização anterior à nova, suavizando a trajetória.
- Se os gradientes recentes apontam na mesma direção, aceleramos.
- Se mudam de direção, o efeito tende a se cancelar e o passo é amortecido.

Momentum[7]

Atualização com momentum

$$v_{t+1} = \beta\, v_t + (1 - \beta)\, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)$$

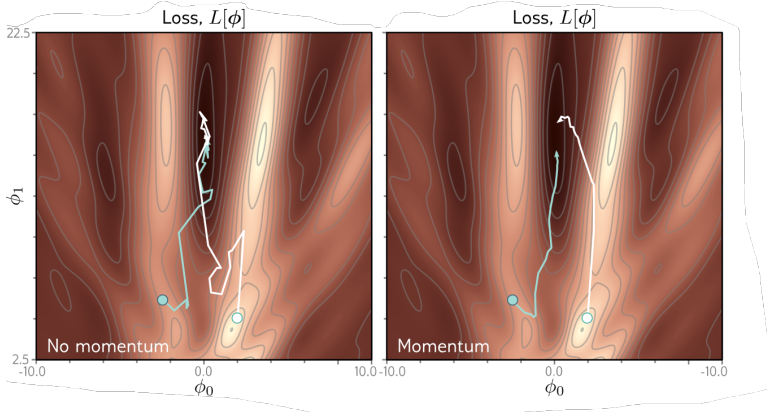
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\, v_{t+1}$$

- $\beta \in [0, 1)$ controla quanto da memória é considerado.
- v_t é uma **média móvel exponencial** dos gradientes anteriores.
- Para $\beta = 0$, recuperamos a descida de gradiente padrão.
- Valores típicos: $\beta \in [0.9\,;\, 0.99]$.

Convenção sem $(1 - \beta)$: PyTorch e o paper original de Polyak escrevem $v_{t+1} = \beta\, v_t + \nabla_{\boldsymbol{\theta}} \mathcal{L}$. A diferença é absorvível em η.

[7] Boris T. Polyak. "Some methods of speeding up the convergence of iteration methods". Em: USSR Computational Mathematics and Mathematical Physics 4.5 (1964), pp. 1–17.

Efeito visual do *Momentum*



- Esquerda: SGD puro oscila fortemente em vales estreitos.
- Direita: com momentum, oscilações verticais se cancelam e a trajetória avança mais rápido na direção principal.

Exemplo numérico: *Momentum*

Cenário

$$\eta = 0.1, \quad \beta = 0.9, \quad \theta_t = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad \nabla_\theta \mathcal{L}(\theta_t) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad v_t = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

$$v_{t+1} = 0.9 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1.9 \\ 1.1 \end{bmatrix}$$

$$\theta_{t+1} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 1.9 \\ 1.1 \end{bmatrix} = \begin{bmatrix} 2.81 \\ 4.89 \end{bmatrix}$$

- A memória v_t empurra o passo além do gradiente atual: 0.19 na primeira coordenada, contra 0.1 do SGD.

Nesterov Accelerated Momentum[8]

- Momentum é, na prática, um *preditor* do próximo passo.
- Nesterov sugere avaliar o gradiente já no ponto deslocado pelo momentum, e não no ponto atual.

Regra de Nesterov

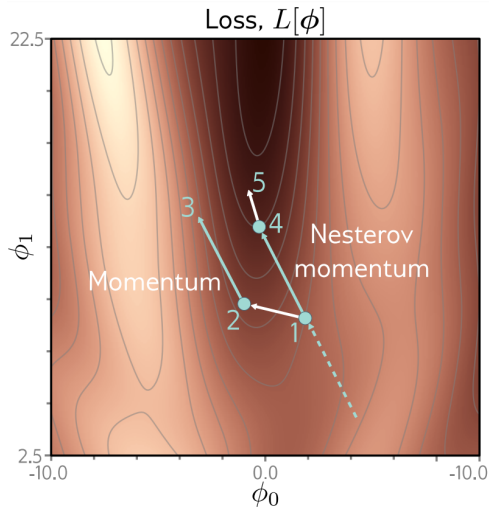
$$v_{t+1} = \beta\, v_t + (1-\beta)\, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t - \eta\, \beta\, v_t)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\, v_{t+1}$$

- Pequena correção, ganho prático em problemas com vales alongados.

[8] Yurii Nesterov. "A method of solving a convex programming problem with convergence rate O($1/k^2$)". Em: Soviet Mathematics Doklady 27.2 (1983), pp. 372–376.

Nesterov: olhar à frente antes de calcular o gradiente



- O ponto branco é onde o momentum levaria, e é nesse ponto que o gradiente é avaliado.
- Resultado: a trajetória do Nesterov "corrige" a do momentum quando há curvatura.

Exemplo numérico: Nesterov

Cenário

$$\eta = 0.1, \quad \beta = 0.9, \quad \theta_t = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad \upsilon_t = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Ponto adiantado pelo momentum e gradiente avaliado nele:

$$\theta_t - \eta\beta\,\upsilon_t = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.09 \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.82 \\ 4.91 \end{bmatrix}, \qquad \nabla_\theta \mathcal{L}\left(\begin{bmatrix} 2.82 \\ 4.91 \end{bmatrix}\right) = \begin{bmatrix} 0.8 \\ 1.6 \end{bmatrix}.$$

$$\upsilon_{t+1} = 0.9 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 0.1 \begin{bmatrix} 0.8 \\ 1.6 \end{bmatrix} = \begin{bmatrix} 1.88 \\ 1.06 \end{bmatrix}$$

$$\theta_{t+1} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 1.88 \\ 1.06 \end{bmatrix} = \begin{bmatrix} 2.812 \\ 4.894 \end{bmatrix}$$

- O gradiente é medido já no ponto deslocado pelo momentum, não em θ_t.

Problema do passo fixo em paisagens anisotrópicas

- Em uma direção a loss pode ser muito íngreme; em outra, quase plana.
- Com η fixo, ou damos passos exagerados na direção íngreme, ou ficamos lentos na direção rasa.
- Solução: usar uma taxa de aprendizado **efetiva** diferente para cada parâmetro, baseada na magnitude histórica de seus gradientes.

Normalização do gradiente

Para tornar o passo independente da magnitude do gradiente, podemos definir

$$m_{t+1} = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t), \qquad v_{t+1} = \left(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)\right)^2$$

e atualizar com

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \, \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon}.$$

- Com a normalização, o termo que multiplica η tem magnitude próxima de 1.
- O tamanho do passo passa a ser controlado essencialmente por η.
- ϵ é um pequeno valor positivo para estabilidade numérica.

Exemplo numérico: normalização do gradiente

Cenário

$$\eta = 0.1, \quad \theta_t = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad \nabla_\theta \mathcal{L}(\theta_t) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \epsilon \approx 0.$$

$$m_{t+1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \qquad v_{t+1} = \begin{bmatrix} 1^2 \\ 2^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

$$\theta_{t+1} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 1/\sqrt{1} \\ 2/\sqrt{4} \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.9 \\ 4.9 \end{bmatrix}.$$

- Divisão elemento a elemento. Apesar do gradiente assimétrico, o passo normalizado é igual nas duas coordenadas.

AdaGrad: *Adaptive (per-parameter) Gradient*[9]

Atualização

$$G_{t+1} = G_t + \left(\nabla_{\boldsymbol{\theta}} \mathcal{L}\right)^2$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{G_{t+1}} + \epsilon} \nabla_{\boldsymbol{\theta}} \mathcal{L}$$

- Cada parâmetro tem sua própria taxa efetiva, baseada em G_t, a soma acumulada dos gradientes ao quadrado.
- Gradientes consistentemente grandes recebem passos menores; parâmetros raros recebem passos maiores.

Limitação

G_t é monotonamente crescente, logo $\eta/(\sqrt{G_t} + \epsilon)$ decai sempre: a taxa efetiva tende a zero e o aprendizado para.

[9] John Duchi, Elad Hazan e Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". Em: JMLR 12 (2011), pp. 2121–2159.

Exemplo numérico: AdaGrad

Cenário

$$\eta = 0.1, \quad \theta_t = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad \nabla_\theta \mathcal{L}(\theta_t) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad G_t = \begin{bmatrix} 24 \\ 12 \end{bmatrix}.$$

$$G_{t+1} = \begin{bmatrix} 24 \\ 12 \end{bmatrix} + \begin{bmatrix} 1^2 \\ 2^2 \end{bmatrix} = \begin{bmatrix} 25 \\ 16 \end{bmatrix}.$$

$$\theta_{t+1} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 1/\sqrt{25} \\ 2/\sqrt{16} \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 0.2 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 2.98 \\ 4.95 \end{bmatrix}.$$

- G_t acumula os quadrados dos gradientes e só cresce. Cada coordenada ganha uma taxa efetiva própria $\eta/\sqrt{G_{t+1}}$, então o passo $(0.02,\ 0.05)$ difere entre os parâmetros: fica menor onde G acumulou mais.

RMSProp: *Root Mean Square Propagation*[10]

- Apresentado por Hinton em um *slide* de aula; nunca foi formalmente publicado.
- Disponível em praticamente todos os *frameworks*.
- Ideia: substituir a soma do AdaGrad por uma **média móvel exponencial**.

Atualização

$$\mathbb{E}[g^2]_{t+1} = \rho\, \mathbb{E}[g^2]_t + (1-\rho)\, (\nabla_{\boldsymbol{\theta}} \mathcal{L})^2$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_{t+1}} + \epsilon} \nabla_{\boldsymbol{\theta}} \mathcal{L}$$

- ρ controla a janela efetiva de memória; o termo deixa de crescer indefinidamente.

[10] Tijmen Tieleman e Geoffrey Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 2012.

Exemplo numérico: RMSProp

Cenário

$$\eta = 0.1, \quad \rho = 0.9, \quad \theta_t = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad \nabla_\theta \mathcal{L}(\theta_t) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \mathbb{E}[g^2]_t = \begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

$$\mathbb{E}[g^2]_{t+1} = 0.9 \begin{bmatrix} 1 \\ 4 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

$$\theta_{t+1} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 1/\sqrt{1} \\ 2/\sqrt{4} \end{bmatrix} = \begin{bmatrix} 2.9 \\ 4.9 \end{bmatrix}.$$

- Neste cenário o passo coincide com o da normalização simples. A diferença para o AdaGrad (média móvel contra soma acumulada) aparece ao longo das iterações.

Adam: *Adaptive Moment Estimation*[11]

- Junta as duas ideias vistas: **momentum** (sobre o gradiente) e **normalização** (pela raiz da média móvel do gradiente ao quadrado).
- Considerado, por muitos anos, o otimizador padrão para deep learning.

[11] Diederik P. Kingma e Jimmy Ba. "Adam: A Method for Stochastic Optimization". Em: ICLR. 2015.

Adam: as duas médias móveis

Estimativas dos momentos

$$m_{t+1} = \beta_1 \, m_t + (1 - \beta_1) \, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)$$

$$v_{t+1} = \beta_2 \, v_t + (1 - \beta_2) \, \left(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t) \right)^2$$

- $\beta_1 \in [0, 1)$, $\beta_2 \in [0, 1)$, com $m_0 = 0$ e $v_0 = 0$.
- m acompanha o primeiro momento (média) do gradiente, v acompanha o segundo momento (média do gradiente ao quadrado).

Atenção à notação: aqui v é a média móvel do gradiente ao quadrado, não a velocidade do *momentum*.

Adam: correção de viés inicial

- Com $m_0 = v_0 = 0$, as primeiras iterações subestimam fortemente m e v.
- Correção de viés (*bias correction*):

$$\widetilde{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^{t+1}}, \qquad \widetilde{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}.$$

- A regra final é

$$\boxed{\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \, \frac{\widetilde{m}_{t+1}}{\sqrt{\widetilde{v}_{t+1}} + \epsilon}}$$

- Hiperparâmetros padrão: β_1=0.9, β_2=0.999, ϵ=10^{-8}.

Exemplo numérico: Adam (primeiro passo)

Cenário

$$\eta = 0.1,\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ \theta_0 = \begin{bmatrix} 3 \\ 5 \end{bmatrix},\ \nabla_\theta \mathcal{L}(\theta_0) = \begin{bmatrix} 1 \\ 2 \end{bmatrix},\ m_0 = v_0 = 0.$$

$$m_1 = 0.1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}, \qquad v_1 = 0.001 \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 0.001 \\ 0.004 \end{bmatrix}.$$

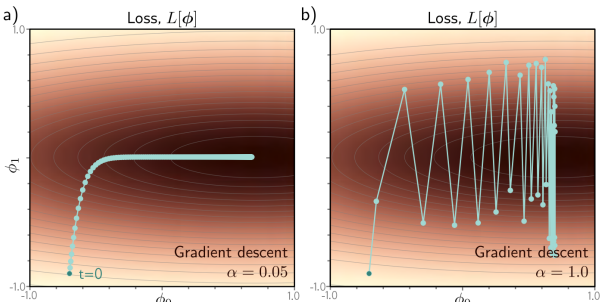
Correção de viés ($t = 1$):

$$\widetilde{m}_1 = \frac{1}{1 - 0.9} \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \qquad \widetilde{v}_1 = \frac{1}{1 - 0.999} \begin{bmatrix} 0.001 \\ 0.004 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

$$\theta_1 = \begin{bmatrix} 3 \\ 5 \end{bmatrix} - 0.1 \begin{bmatrix} 1/\sqrt{1} \\ 2/\sqrt{4} \end{bmatrix} = \begin{bmatrix} 2.9 \\ 4.9 \end{bmatrix}.$$

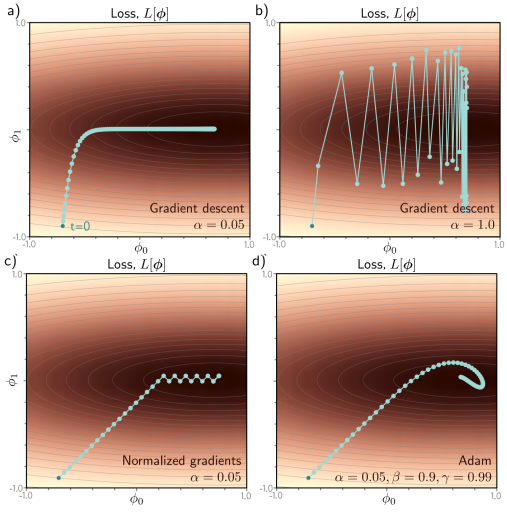
- A correção de viés desfaz a subestimativa inicial: o passo por parâmetro fica $\approx \eta$.

Por que o Adam ajuda em loss anisotrópica



- Esquerda: η pequeno avança bem na vertical, mas é lento na horizontal.
- Direita: η grande progride na horizontal, mas oscila descontroladamente na vertical.
- Adam normaliza por direção, evitando ambos os extremos.

Adam comparado com outros otimizadores



Por que Adam virou o padrão

- As magnitudes dos gradientes variam fortemente conforme a função de ativação, a camada da rede e a inicialização.
- Adam é mais imune a essas diferenças e menos sensível à escolha da taxa de aprendizado inicial.
- Na prática, costuma funcionar razoavelmente bem com os hiperparâmetros padrão.

AdamW: *weight decay* desacoplado[12]

- *Weight decay*: encolher os pesos um pouco a cada passo, penalizando pesos de magnitude alta.
- No SGD, basta somar $\lambda\,\boldsymbol{\theta}_t$ ao gradiente: o efeito é o decaimento multiplicativo $(1-\eta\lambda)\,\boldsymbol{\theta}_t$.
- No Adam, esse termo entraria em m e v e passaria pela divisão por $\sqrt{\widetilde{v}}$: pesos com histórico de gradientes grandes quase não decairiam.
- O **AdamW** desacopla o decaimento, aplicando-o fora da parte adaptativa:

Atualização do AdamW

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\left(\frac{\widetilde{m}_{t+1}}{\sqrt{\widetilde{v}_{t+1}}+\epsilon} + \lambda\,\boldsymbol{\theta}_t\right)$$

[12] Ilya Loshchilov e Frank Hutter. "Decoupled weight decay regularization". Em: ICLR. 2019.

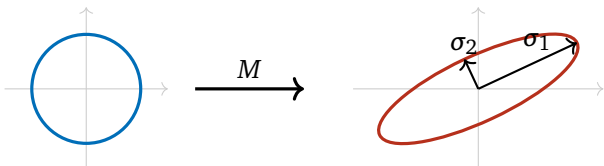
Muon: *MomentUm Orthogonalized by Newton-schulz*[13]

- Por quase uma década, Adam (e variantes como AdamW) foi o padrão de fato em deep learning.
- Em 2024, o **Muon** foi proposto, apresentado pelos autores como o primeiro avanço fundamental desde Adam.
- Foco do Muon: as matrizes de pesos das camadas ocultas (parâmetros 2D). Bias e *embeddings* continuam usando AdamW.

[13] Keller Jordan et al. Muon: An optimizer for hidden layers in neural networks. Blog post. 2024. URL: https://kellerjordan.github.io/posts/muon/.

Valores singulares: os fatores de esticamento da matriz

Uma matriz M transforma o círculo unitário, o conjunto dos vetores de norma 1, em uma elipse.

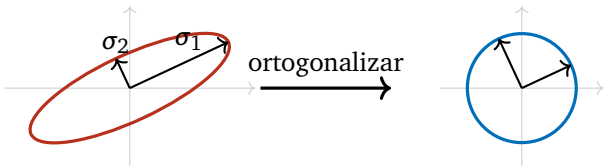


vetores de norma 1 a imagem é uma elipse

- Os **valores singulares** $\sigma_1 \geq \sigma_2$ são os comprimentos dos semieixos.
- Vetores ao longo da direção de σ grande saem muito amplificados; ao longo da de σ pequeno, quase apagados.

Ortogonalizar: transformar a elipse em círculo

Uma matriz O é **ortogonal** quando $O^\top O = I$, ou seja, suas colunas têm norma 1 e são ortogonais entre si.



- Uma matriz ortogonal preserva comprimentos e ângulos: o círculo unitário continua um círculo unitário.
- **Ortogonalizar** M é trocá-la pela matriz ortogonal mais próxima, mantendo as direções singulares e igualando todos os σ a 1.

Ideia central: ortogonalizar a atualização

- As matrizes de gradientes em redes neurais costumam ter *rank efetivo* baixo, ou seja, são dominadas por poucas direções singulares.
- O passo fica mal aproveitado: direções com σ pequeno, mesmo quando úteis para o aprendizado, quase não contribuem para a atualização.
- Muon substitui a matriz de atualização pela matriz ortogonal mais próxima, redistribuindo a magnitude entre todas as direções singulares.
- Geometricamente, o passo do Muon é a *steepest descent* sob a norma espectral, enquanto o SGD é a *steepest descent* sob a norma de Frobenius.

Steepest descent: o passo mais íngreme depende da norma

Qual atualização ΔW mais reduz a loss, dado um tamanho máximo de passo? Com $G = \nabla_W L$, a aproximação linear da loss é

$$L(W + \Delta W) \;\approx\; L(W) + \langle G,\, \Delta W \rangle, \qquad \langle G,\, \Delta W \rangle = \textstyle\sum_{ij} G_{ij}\, \Delta W_{ij}.$$

O passo mais íngreme minimiza essa aproximação com o tamanho do passo limitado:

$$\Delta W^* = \arg\min_{\|\Delta W\| \le \eta} \langle G,\, \Delta W \rangle.$$

- A restrição é essencial: na aproximação linear, sem limite de tamanho o valor desceria sem fim.
- A solução depende de qual norma mede o tamanho do passo. Vamos resolver o problema para duas normas: a de **Frobenius** e a **espectral**.

Solução sob a norma de Frobenius: o gradiente

A norma de Frobenius trata a matriz como um vetor longo: $\|\Delta W\|_F = \sqrt{\sum_{ij} \Delta W_{ij}^2}$. Pela desigualdade de Cauchy–Schwarz,

$$\langle G,\, \Delta W \rangle \;\geq\; -\,\|G\|_F\,\|\Delta W\|_F \;\geq\; -\,\eta\,\|G\|_F.$$

A igualdade acontece quando ΔW aponta no sentido oposto a G e usa todo o tamanho permitido:

$$\Delta W^* = -\,\eta\,\frac{G}{\|G\|_F}.$$

- O passo mais íngreme sob Frobenius é o gradiente (normalizado): a descida de gradiente usual mede o tamanho do passo entrada por entrada, sem olhar a estrutura da matriz.

Solução sob a norma espectral: a ortogonalização

A SVD escreve G como uma soma de matrizes de *rank* 1: cada parcela pega a direção singular de entrada v_i, leva na direção de saída u_i e estica por σ_i:

$$G = \sum_i \sigma_i \, u_i v_i^\top \qquad \Longrightarrow \qquad \langle G,\, \Delta W \rangle = \sum_i \sigma_i \left(u_i^\top \Delta W \, v_i \right).$$

A norma espectral é o maior fator de esticamento: se $\|\Delta W\|_2 = \sigma_1(\Delta W) \le \eta$, nenhum vetor unitário é esticado além de η. Cada parcela então obedece $u_i^\top \Delta W \, v_i \ge -\eta$, logo

$$\langle G,\, \Delta W \rangle \;\ge\; -\eta \sum_i \sigma_i.$$

A igualdade exige $\Delta W \, v_i = -\eta \, u_i$ em todas as direções singulares ao mesmo tempo:

$$\Delta W^* = -\eta \sum_i u_i v_i^\top = -\eta \, U V^\top,$$

uma matriz ortogonal escalada por η, exatamente a ortogonalização de G.

Supomos todos os $\sigma_i > 0$; U e V empilham os u_i e v_i como colunas; usamos $\langle u_i v_i^\top,\, \Delta W \rangle = u_i^\top \Delta W \, v_i$ e $\|u_i\| = \|v_i\| = 1$.

Pseudocódigo do Muon

Atualização (matriz G do gradiente)

1: $M_{t+1} \leftarrow \beta M_t + G_t$ ▷ momentum (acumulador de gradientes)
2: $\widehat{M}_{t+1} \leftarrow M_{t+1} / \|M_{t+1}\|_F$ ▷ normaliza pela norma de Frobenius
3: $O_{t+1} \leftarrow \text{NewtonSchulz5}(\widehat{M}_{t+1})$ ▷ aproxima a matriz ortogonal mais próxima
4: $W_{t+1} \leftarrow W_t - \eta\, O_{t+1}$

- Newton-Schulz aproxima a parte ortogonal de $\widehat{M}$ com polinômios matriciais, sem calcular SVD; com 5 passos, roda bem em *tensor cores* de GPU.
- A normalização da linha 2 existe só para trazer os valores singulares à região de convergência do Newton-Schulz: a matriz ortogonal mais próxima não muda com a escala.
- Quem é ortogonalizado é o momentum M_{t+1}, não o gradiente instantâneo: na derivação do passo mais íngreme, M_{t+1} assume o papel de G.

Na implementação de referência, a normalização fica dentro do `NewtonSchulz5` e o momentum é Nesterov; o passo de uma matriz $m \times n$ é reescalado por $\sqrt{\max(1,\, m/n)}$, e o Moonlight usa $0.2\sqrt{\max(m, n)}$ para casar com o AdamW.

Exemplo numérico: Muon (momentum e normalização)

Cenário: parâmetro 2D, como o Muon exige

$$\eta = 0.1, \quad \beta = 0.9, \quad W_t = \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}, \quad M_t = \begin{bmatrix} 2 & -3 \\ 5 & 2 \end{bmatrix}, \quad G_t = \begin{bmatrix} 1.2 & 1.7 \\ 2.5 & 1.2 \end{bmatrix}.$$

Passo 1, momentum (elemento a elemento):

$$M_{t+1} = 0.9 \begin{bmatrix} 2 & -3 \\ 5 & 2 \end{bmatrix} + \begin{bmatrix} 1.2 & 1.7 \\ 2.5 & 1.2 \end{bmatrix} = \begin{bmatrix} 3 & -1 \\ 7 & 3 \end{bmatrix}.$$

Passo 2, normalização pela norma de Frobenius:

$$\|M_{t+1}\|_F = \sqrt{9 + 1 + 49 + 9} = \sqrt{68} \approx 8.25, \qquad \widehat{M}_{t+1} \approx \begin{bmatrix} 0.36 & -0.12 \\ 0.85 & 0.36 \end{bmatrix}.$$

- Falta o passo 3, a ortogonalização. Antes dele, vejamos quanto M_{t+1} estica cada direção singular.

Os valores singulares de M_{t+1}

Os valores singulares de M são as raízes quadradas dos autovalores de $M^\top M$:

$$M_{t+1}^\top M_{t+1} = \begin{bmatrix} 3 & 7 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 7 & 3 \end{bmatrix} = \begin{bmatrix} 58 & 18 \\ 18 & 10 \end{bmatrix}.$$

Com traço e determinante

$$\operatorname{tr}(M^\top M) = 58 + 10 = 68, \qquad \det(M^\top M) = 58 \cdot 10 - 18 \cdot 18 = 580 - 324 = 256,$$

a equação característica fica

$$\lambda^2 - 68\lambda + 256 = 0, \qquad \lambda = \frac{68 \pm \sqrt{4624 - 1024}}{2} = \frac{68 \pm 60}{2}.$$

$$\lambda_1 = 64, \quad \lambda_2 = 4 \qquad \Longrightarrow \qquad \sigma_1 = 8, \quad \sigma_2 = 2.$$

- Vetores em uma direção singular saem amplificados 4 vezes mais que na outra: é essa assimetria que o Muon vai corrigir.

Exemplo numérico: Muon (a tentativa ingênua)

As colunas de $M_{t+1} = \begin{bmatrix} 3 & -1 \\ 7 & 3 \end{bmatrix}$ não são ortogonais entre si:

$$(3,\, 7) \cdot (-1,\, 3) = -3 + 21 = 18 \neq 0.$$

A tentativa ingênua seria normalizar cada coluna separadamente:

$$\frac{(3,\, 7)}{\sqrt{58}} = (0.39,\, 0.92), \qquad \frac{(-1,\, 3)}{\sqrt{10}} = (-0.32,\, 0.95).$$

Normalizar as colunas não basta

Cada coluna ficou com norma 1, mas o produto interno entre elas é 0.75, ainda longe de zero. A matriz resultante não é ortogonal.

Exemplo numérico: Muon (ortogonalização e atualização)

Passo 3, ortogonalização de $M_{t+1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ por forma fechada (requer $\det M_{t+1} > 0$):

$$\det M_{t+1} = 3 \cdot 3 - (-1) \cdot 7 = 16, \qquad p = a + d = 6, \qquad q = c - b = 8, \qquad \sqrt{p^2 + q^2} = 10,$$

$$O_{t+1} = \frac{1}{10} \begin{bmatrix} 6 & -8 \\ 8 & 6 \end{bmatrix} = \begin{bmatrix} 0.6 & -0.8 \\ 0.8 & 0.6 \end{bmatrix}.$$

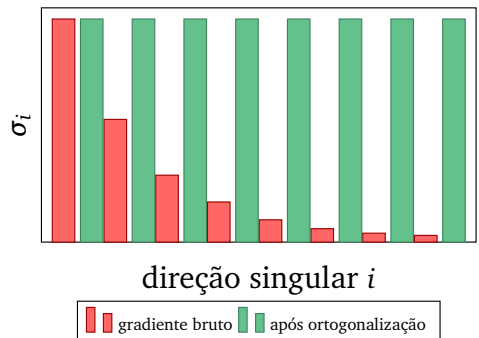
Passo 4, atualização dos pesos:

$$W_{t+1} = \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix} - 0.1 \begin{bmatrix} 0.6 & -0.8 \\ 0.8 & 0.6 \end{bmatrix} = \begin{bmatrix} 2.94 & 5.08 \\ 1.92 & 3.94 \end{bmatrix}.$$

- Os valores singulares passaram de $(8, 2)$ para $(1, 1)$: a atualização estica todas as direções singulares igualmente.
- Aplicamos a forma fechada a M_{t+1}, não a $\widehat{M}_{t+1}$: as duas têm a mesma ortogonal mais próxima. A normalização do passo 2 só importa para a convergência do Newton-Schulz.

Intuição: por que ortogonalizar ajuda?

- Gradiente típico: poucas direções com magnitude muito alta, muitas direções com magnitude pequena.
- Adam normaliza *por parâmetro* (entrada da matriz), mas não trata o problema de *rank* efetivo.
- Muon normaliza *por direção singular*: todas as direções recebem magnitude semelhante.
- Resultado: melhor uso da capacidade de cada passo, especialmente em redes largas.



Resultados empíricos

- Recordes de velocidade em *NanoGPT speedrun* (~1.35× speedup para atingir loss de validação 3.28).
- Recordes em CIFAR-10 *speedrun* (de 3.3 para 2.6 segundos-A100 para atingir 94% de acurácia).
- Em larga escala: o Moonlight[14], um MoE de 16B de parâmetros, foi treinado com Muon; o Kimi K2[15], com 1 trilhão de parâmetros, usou a variante MuonClip.

Limitações práticas

Aplica-se apenas a parâmetros 2D (matrizes). Vetores (*biases*, *layer norm*) e *embeddings* continuam sendo otimizados com AdamW.

[14] Jingyuan Liu et al. "Muon is scalable for LLM training". Em: arXiv preprint arXiv:2502.16982 (2025).

[15] Kimi Team. "Kimi K2: Open agentic intelligence". Em: arXiv preprint arXiv:2507.20534 (2025).

Visão Geral

Qual otimizador usar na prática?

- **Padrão hoje**: AdamW (Adam com *weight decay* desacoplado) na maioria dos modelos.
- **Visão computacional clássica**: SGD com momentum costuma generalizar tão bem ou melhor, especialmente com *learning rate schedules* bem ajustados.
- **Modelos grandes (LLMs)**: AdamW continua sendo o padrão; Muon vem ganhando adoção para as matrizes das camadas ocultas (Moonlight, Kimi K2).
- **Importante**: a taxa de aprendizado η é o hiperparâmetro de maior impacto em qualquer um deles.

Leituras Recomendadas

- Capítulos 4 e 8 de Ian Goodfellow, Yoshua Bengio e Aaron Courville. Deep Learning. MIT Press, 2016. URL: https://www.deeplearningbook.org/
- Capítulo 6 de Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023
- Adam original: Diederik P. Kingma e Jimmy Ba. “Adam: A Method for Stochastic Optimization”. Em: ICLR. 2015
- Muon: Keller Jordan et al. Muon: An optimizer for hidden layers in neural networks. Blog post. 2024. URL: https://kellerjordan.github.io/posts/muon/

Referências I

[1] John Duchi, Elad Hazan e Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. Em: JMLR 12 (2011), pp. 2121–2159.

[2] Ian Goodfellow, Yoshua Bengio e Aaron Courville. Deep Learning. MIT Press, 2016. URL: https://www.deeplearningbook.org/.

[3] Keller Jordan et al. Muon: An optimizer for hidden layers in neural networks. Blog post. 2024. URL: https://kellerjordan.github.io/posts/muon/.

[4] Kimi Team. “Kimi K2: Open agentic intelligence”. Em: arXiv preprint arXiv:2507.20534 (2025).

[5] Diederik P. Kingma e Jimmy Ba. “Adam: A Method for Stochastic Optimization”. Em: ICLR. 2015.

[6] Jingyuan Liu et al. “Muon is scalable for LLM training”. Em: arXiv preprint arXiv:2502.16982 (2025).

[7] Ilya Loshchilov e Frank Hutter. “Decoupled weight decay regularization”. Em: ICLR. 2019.

[8] Yurii Nesterov. “A method of solving a convex programming problem with convergence rate $O(1/k^2)$”. Em: Soviet Mathematics Doklady 27.2 (1983), pp. 372–376.

[9] Boris T. Polyak. “Some methods of speeding up the convergence of iteration methods”. Em: USSR Computational Mathematics and Mathematical Physics 4.5 (1964), pp. 1–17.

[10] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.

[11] Tijmen Tieleman e Geoffrey Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 2012.