Inteligência Artificial 2

A Mecânica do Aprendizado (Gradient Descent)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

18 de agosto de 2026

Onde Paramos?

Na aula anterior, construímos a arquitetura MLP e dominamos o **Forward Propagation**:

- Camadas Ocultas, ReLU e Sigmoid.
- A cascata $Z = X \cdot W + b$ seguida de ativação.
- Processamento em batch (múltiplas amostras de uma vez).

A Pergunta de Hoje

A rede propaga, mas os pesos são **aleatórios**.
Como ensiná-la a **aprender**?

Objetivos da Sessão

Ao final desta aula, você será capaz de:

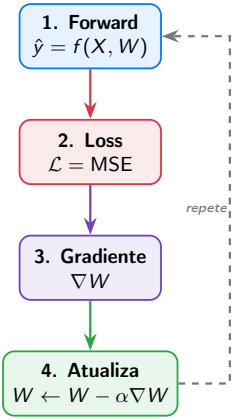
1. **Quantificar o erro** de uma rede usando a Função de Perda (MSE).
2. **Calcular derivadas parciais** e entender o conceito de gradiente.
3. **Executar na mão** a regra de atualização $W_{novo} = W - \alpha \nabla W$.
4. **Explicar** o Backpropagation e a Regra da Cadeia.
5. **Diagnosticar** Exploding e Vanishing Gradients.
6. **Comparar** otimizadores: SGD, Mini-Batch e Adam.

Pré-requisito

Vocês viram derivadas em Cálculo I — hoje elas ganham um propósito real.

O Problema: Pesos Aleatórios = Predições Aleatórias

- Os pesos W são inicializados via `np.random.randn`.
- Na época $t = 0$, a rede **chuta** — suas predições são ruído puro.
- Precisamos de um mecanismo que:
  1. **Meça** o erro.
  2. **Calcule** a direção de correção.
  3. **Ajuste** os pesos automaticamente.



Medindo o Erro: Função de Perda (MSE)

A máquina não tem intuição. Ela precisa de um **número** que diga o quão errada está.

Média dos Erros Quadráticos (Mean Squared Error)

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Medindo o Erro: Função de Perda (MSE)

A máquina não tem intuição. Ela precisa de um **número** que diga o quão errada está.

Média dos Erros Quadráticos (Mean Squared Error)

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- O **quadrado** impede que erros negativos e positivos se anulem.

Medindo o Erro: Função de Perda (MSE)

A máquina não tem intuição. Ela precisa de um **número** que diga o quão errada está.

Média dos Erros Quadráticos (Mean Squared Error)

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- O **quadrado** impede que erros negativos e positivos se anulem.
- A punição é **exponencial**: erro de 10 gera punição 100; erro de 100 gera punição 10.000.

Medindo o Erro: Função de Perda (MSE)

A máquina não tem intuição. Ela precisa de um **número** que diga o quão errada está.

Média dos Erros Quadráticos (Mean Squared Error)

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- O **quadrado** impede que erros negativos e positivos se anulem.
- A punição é **exponencial**: erro de 10 gera punição 100; erro de 100 gera punição 10.000.
- **Objetivo:** minimizar $\mathcal{L}$ até que $\hat{y} \approx y$.

MSE: Exemplo Numérico com Batch

Imagine que nossa MLP do Lab 02 recebeu um **batch de 3 candidatos**:

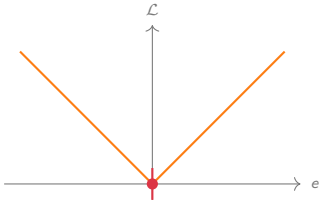
Candidato	y_{real}	$\hat{y}_{predito}$	$(y - \hat{y})^2$
Ana	1.0	0.7	$(0.3)^2 = 0.09$
Bruno	0.0	0.4	$(-0.4)^2 = 0.16$
Carlos	1.0	0.9	$(0.1)^2 = 0.01$

$$\mathcal{L} = \frac{0.09 + 0.16 + 0.01}{3} = \frac{0.26}{3} = 0.0867$$

Pergunta pro professor: Se a rede melhorar a predição de Bruno de 0.4 para 0.1, quanto cai o MSE?
(Faça a conta ao vivo!)

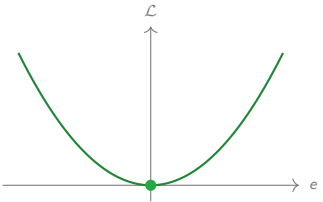
Por que o Quadrado e não o Valor Absoluto?

Erro Absoluto $|y - \hat{y}|$



Não-diferenciável em $e = 0$!

Erro Quadrático $(y - \hat{y})^2$



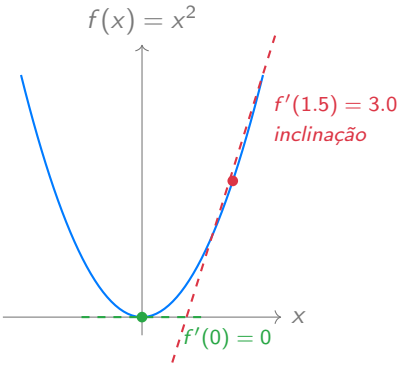
Suave e diferenciável ✓

Conclusão

O quadrado é **diferenciável em todo ponto** — e o Gradient Descent **precisa** de derivadas para funcionar. Sem derivada, sem direção; sem direção, sem aprendizado.

Revisão Relâmpago: O que é uma Derivada?

- A derivada $\frac{df}{dx}$ mede a **taxa de variação** de f quando perturbamos x.
- Se $f(x) = x^2$, então $f'(x) = 2x$.
- **No ponto** $x = 3$: $f'(3) = 6$.
  Significado: "se eu aumentar x um pouquinho, f aumenta ≈ 6 vezes mais rápido".



Conexão com IA

Em IA, x é um **peso** e f é a **perda**. A derivada nos diz para **qual lado** ajustar o peso para reduzir o erro.

Resgatando Cálculo I e III: A Matemática da IA

Cálculo I: Tangente & Sentido

- $f'(x) > 0$: a curva **subindo** $\rightarrow$ andar para a esquerda reduz f.
- $f'(x) < 0$: a curva **descendo** $\rightarrow$ andar para a direita reduz f.
- Em IA: nos diz se aumentar o peso w faz o erro subir ou descer.

Cálculo III: Vetor Gradiente ($\nabla \mathcal{L}$)

- Vetor com todas as derivadas parciais: $\nabla \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \ldots, \frac{\partial \mathcal{L}}{\partial w_n} \right]^T$
- **Teorema:** $\nabla \mathcal{L}$ aponta para a **maior subida**.
- **Descida Mais Íngreme:** caminhamos em $-\nabla \mathcal{L}$.

Conclusão Fundamental

Para minimizar a perda, a regra de atualização é sempre: $W_{novo} = W - \alpha \cdot \nabla W$.

Derivada Parcial: Múltiplos Pesos

Na prática, uma rede tem **milhares de pesos**. Precisamos de derivadas parciais:

Exemplo com 2 Pesos

Função de perda: $\mathcal{L}(w_1, w_2) = w_1^2 + w_2^2$

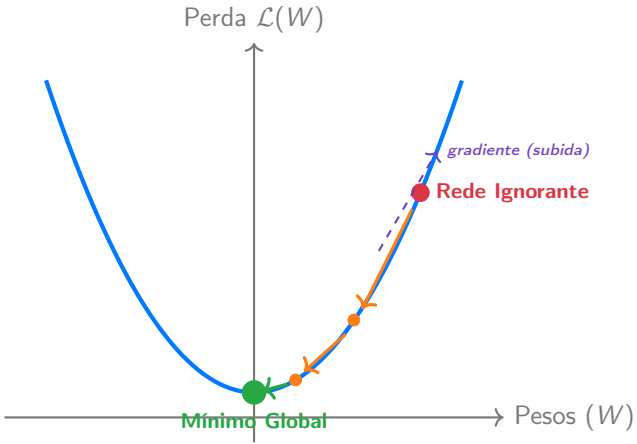
$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1 \qquad \frac{\partial \mathcal{L}}{\partial w_2} = 2w_2$$

- $\frac{\partial \mathcal{L}}{\partial w_1}$: "quanto a perda muda se eu mexer **apenas** em w_1, mantendo w_2 fixo?"
- O **vetor gradiente** $\nabla \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}\right]$ aponta para a direção de **maior subida**.

Intuição

Cada peso tem sua "culpa" no erro. A derivada parcial mede exatamente **quanto cada peso é culpado**.

A Metáfora do Alpinista Vendado



O alpinista **sente a inclinação com o pé** (derivada), e caminha **contra a subida**!

Gradient Descent: A Regra de Atualização

O coração de **todo** aprendizado de máquina cabe em uma linha:

Update Rule

$$\underbrace{W_{novo}}_{\text{peso corrigido}} = \underbrace{W_{atual}}_{\text{peso velho}} - \underbrace{\alpha}_{\text{tamanho do passo}} \cdot \underbrace{\nabla W}_{\text{direção de subida}}$$

Gradient Descent: A Regra de Atualização

O coração de **todo** aprendizado de máquina cabe em uma linha:

Update Rule

$$\underbrace{W_{novo}}_{\text{peso corrigido}} = \underbrace{W_{atual}}_{\text{peso velho}} - \underbrace{\alpha}_{\text{tamanho do passo}} \cdot \underbrace{\nabla W}_{\text{direção de subida}}$$

- ∇W = gradiente = aponta para a **subida** mais íngreme.

Gradient Descent: A Regra de Atualização

O coração de **todo** aprendizado de máquina cabe em uma linha:

Update Rule

$$\underbrace{W_{novo}}_{\text{peso corrigido}} = \underbrace{W_{atual}}_{\text{peso velho}} - \underbrace{\alpha}_{\text{tamanho do passo}} \cdot \underbrace{\nabla W}_{\text{direção de subida}}$$

- ∇W = gradiente = aponta para a **subida** mais íngreme.
- Subtraímos porque queremos a **descida** (o sinal de menos inverte a direção!).

Gradient Descent: A Regra de Atualização

O coração de **todo** aprendizado de máquina cabe em uma linha:

Update Rule

$$\underbrace{W_{novo}}_{\text{peso corrigido}} = \underbrace{W_{atual}}_{\text{peso velho}} - \underbrace{\alpha}_{\text{tamanho do passo}} \cdot \underbrace{\nabla W}_{\text{direção de subida}}$$

- ∇W = gradiente = aponta para a **subida** mais íngreme.
- Subtraímos porque queremos a **descida** (o sinal de menos inverte a direção!).
- α = **Learning Rate** (Taxa de Aprendizado) = tamanho do passo.

Gradient Descent: A Regra de Atualização

O coração de **todo** aprendizado de máquina cabe em uma linha:

Update Rule

$$\underbrace{W_{novo}}_{\text{peso corrigido}} = \underbrace{W_{atual}}_{\text{peso velho}} - \underbrace{\alpha}_{\text{tamanho do passo}} \cdot \underbrace{\nabla W}_{\text{direção de subida}}$$

- ∇W = gradiente = aponta para a **subida** mais íngreme.
- Subtraímos porque queremos a **descida** (o sinal de menos inverte a direção!).
- α = **Learning Rate** (Taxa de Aprendizado) = tamanho do passo.

Analogia

É como ajustar o volume do rádio: se está muito alto (erro grande), você gira o botão (peso) um pouco para baixo. O Learning Rate define **quanto** você gira a cada tentativa.

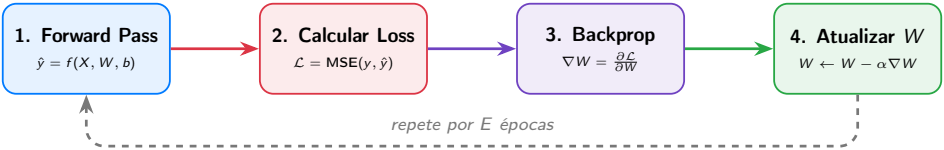
Passo-a-Passo Numérico: 5 Épocas de Treino

Função de perda simplificada: $\mathcal{L}(W) = W^2$, derivada: $\nabla W = 2W$.
Início: $W_0 = 10.0$, Learning Rate: $\alpha = 0.1$.

Época	W	$\mathcal{L} = W^2$	$\nabla W = 2W$	$W_{novo} = W - 0.1 \cdot \nabla W$
0	10.0	100.0	20.0	$10 - 2.0 = 8.0$
1	8.0	64.0	16.0	$8 - 1.6 = 6.4$
2	6.4	40.96	12.8	$6.4 - 1.28 = 5.12$
3	5.12	26.21	10.24	$5.12 - 1.024 = 4.096$
4	4.096	16.78	8.192	$4.096 - 0.819 = 3.277$
	⋮	(continua descendo...)		
20	≈ 0.12	≈ 0.015	≈ 0.24	≈ 0.10

Observe: o erro caiu de **100.0** para **0.015** em 20 épocas. A rede **aprendeu**!

O que Acontece Internamente a Cada Época?



Cada **volta completa** neste ciclo é uma **época** (*epoch*). A rede treina centenas ou milhares de épocas até convergir.

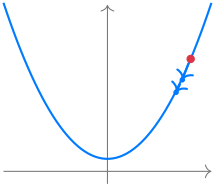
Em Python

```
for epoch in range(epochs):
```

← *Este* é o laço que vocês vão implementar no lab.

Learning Rate: O Tamanho do Passo

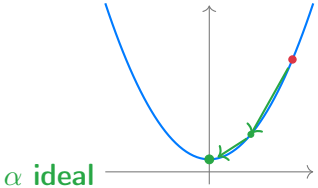
α **muito pequeno**



Convergência lenta

Passos de formiga

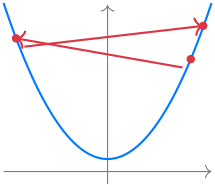
α **ideal**



Convergência suave

Eficiente

α **muito grande**



Divergência!

Quica para fora

Metáfora II: O Chuveiro e o Learning Rate (α)

Calibrando a Temperatura no Inverno

Ajustar o Learning Rate (α) é idêntico a tentar acertar a água do chuveiro num dia frio:

α muito pequeno

Giro de 0.1 mm

- Leva 20 min congelando no banho.
- Treino lentíssimo e custoso.

α muito grande

Giro violento de 180°

- Água ferve e queima as costas!
- Quica de lado $\rightarrow$ `NaN` (*Exploding*).

α ideal

Giro firme e calibrado

- Esquenta em segundos com estabilidade.
- Convergência rápida e segura.

A Catástrofe: Exploding Gradient

O que acontece se α for colossal (ex: 10.0)?

Época	W	$\mathcal{L} = W^2$	$W_{novo} = W - 10 \cdot 2W$
0	10.0	100	$10 - 200 = -190$
1	-190.0	36 100	$-190 + 3\,800 = 3\,610$
2	3 610	$\approx 13 \times 10^6$	$\rightarrow -68\,590$
3	$-68\,590$	$\approx 4.7 \times 10^9$	$\rightarrow$ `inf` $\rightarrow$ **NaN**

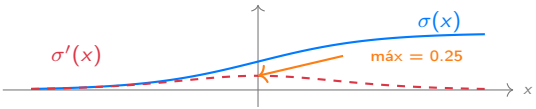
Regra de Sobrevivência

Se o seu treino retornar `NaN`: **reduza o Learning Rate**. Comece com $\alpha = 0.001$ e suba devagar.

O Irmão Silencioso: Vanishing Gradient

O oposto do Exploding: os gradientes ficam tão **pequenos** que os pesos praticamente param de atualizar.

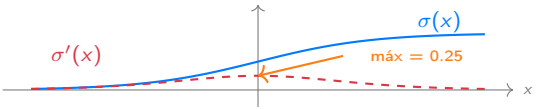
- **O Grito e a Tubulação:** Para o grito de erro voltar, ele passa pela *derivada*. A **Sigmoid** é um tubo enferrujado!



O Irmão Silencioso: Vanishing Gradient

O oposto do Exploding: os gradientes ficam tão **pequenos** que os pesos praticamente param de atualizar.

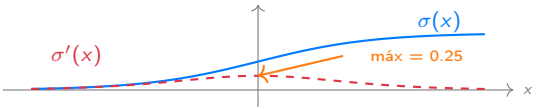
- **O Grito e a Tubulação:** Para o grito de erro voltar, ele passa pela *derivada*. A **Sigmoid** é um tubo enferrujado!
- Máximo da derivada da Sigmoid é **0.25**. Nada sai maior que 25%!



O Irmão Silencioso: Vanishing Gradient

O oposto do Exploding: os gradientes ficam tão **pequenos** que os pesos praticamente param de atualizar.

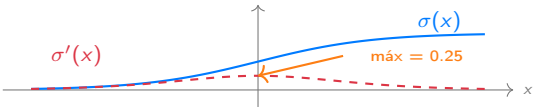
- **O Grito e a Tubulação:** Para o grito de erro voltar, ele passa pela *derivada*. A **Sigmoid** é um tubo enferrujado!
- Máximo da derivada da Sigmoid é **0.25**. Nada sai maior que 25%!
- No Backprop, multiplicamos **em cadeia**: $0.25 \times 0.25 \times 0.25 = 0.016$



O Irmão Silencioso: Vanishing Gradient

O oposto do Exploding: os gradientes ficam tão **pequenos** que os pesos praticamente param de atualizar.

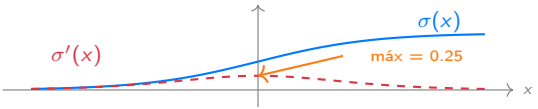
- **O Grito e a Tubulação:** Para o grito de erro voltar, ele passa pela *derivada*. A **Sigmoid** é um tubo enferrujado!
- Máximo da derivada da Sigmoid é **0.25**. Nada sai maior que 25%!
- No Backprop, multiplicamos **em cadeia**: $0.25 \times 0.25 \times 0.25 = 0.016$
- O erro (100) em 5 camadas Sigmoids vira pó (0.09): a rede congela!



O Irmão Silencioso: Vanishing Gradient

O oposto do Exploding: os gradientes ficam tão **pequenos** que os pesos praticamente param de atualizar.

- **O Grito e a Tubulação:** Para o grito de erro voltar, ele passa pela *derivada*. A **Sigmoid** é um tubo enferrujado!
- Máximo da derivada da Sigmoid é **0.25**. Nada sai maior que 25%!
- No Backprop, multiplicamos **em cadeia**: $0.25 \times 0.25 \times 0.25 = 0.016$
- O erro (100) em 5 camadas Sigmoids vira pó (0.09): a rede congela!

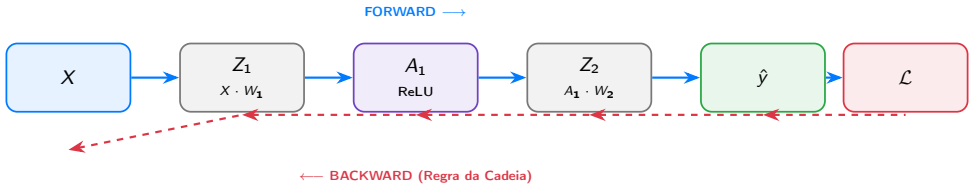


A Solução Definitiva: A Tubulação Perfeita da ReLU

$\text{ReLU}'(x) = 1$ para $x > 0$ — como multiplicar por 1 não diminui o valor original, o gradiente

Backpropagation: A Regra da Cadeia

Como o computador calcula ∇W para pesos em camadas profundas?



$$\frac{\partial \mathcal{L}}{\partial W_1} = \underbrace{\frac{\partial \mathcal{L}}{\partial \hat{y}}}_{\text{loss}} \cdot \underbrace{\frac{\partial \hat{y}}{\partial Z_2}}_{\text{ativação}} \cdot \underbrace{\frac{\partial Z_2}{\partial A_1}}_{\text{peso}} \cdot \underbrace{\frac{\partial A_1}{\partial Z_1}}_{\text{ReLU}} \cdot \underbrace{\frac{\partial Z_1}{\partial W_1}}_{\text{entrada}}$$

Backprop: Exemplo Numérico Simplificado

Considere um neurônio simples: $\hat{y} = \sigma(w \cdot x + b)$, com $\mathcal{L} = (\hat{y} - y)^2$.

Dados

$x = 2.0,\ w = 0.5,\ b = 0.1,\ y_{real} = 1.0$

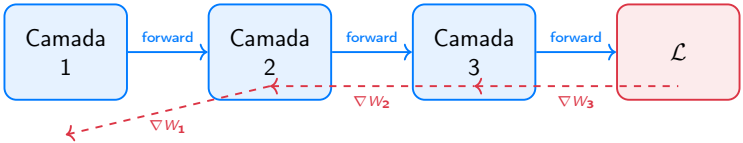
Passo	**Cálculo**	**Valor**
Forward: z	$w \cdot x + b = 0.5 \cdot 2 + 0.1$	1.1
Forward: $\hat{y}$	$\sigma(1.1) = \frac{1}{1+e^{-1.1}}$	0.7502
Loss	$(\hat{y} - y)^2 = (0.75 - 1)^2$	0.0625
$\frac{\partial \mathcal{L}}{\partial \hat{y}}$	$2(\hat{y} - y) = 2(0.75 - 1)$	-0.50
$\frac{\partial \hat{y}}{\partial z}$	$\sigma(z)(1 - \sigma(z)) = 0.75 \cdot 0.25$	0.1875
$\frac{\partial z}{\partial w}$	x	2.0
$\frac{\partial \mathcal{L}}{\partial w}$	$(-0.50) \times 0.1875 \times 2.0$	-0.1875

Como ler a Tabela Numérica?

1. **Forward Pass (A Ida)**
   - z: A conta bruta ($w \cdot x + b$). Deu 1.1.
   - $\hat{y}$: A previsão espremida pela Sigmoid. Deu 0.75.
   - $\mathcal{L}$: O Erro Quadrático, comparando 0.75 com a meta 1.0.
2. **Backward Pass (A Volta)**
   - $\frac{\partial \mathcal{L}}{\partial \hat{y}}$: O Erro da Previsão. Negativo (-0.5) $\rightarrow$ "precisamos subir!".
   - $\frac{\partial \hat{y}}{\partial z}$: O "Tubo" Sigmoid. A inclinação naquele ponto (0.1875).
   - $\frac{\partial z}{\partial w}$: O peso que o dado $x = 2.0$ teve na conta.

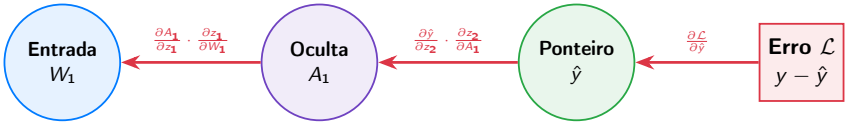
3. Gradiente e 4. Update Rule Multiplica os três passos da volta em cadeia para achar a **Culpa Final** (-0.1875) e atualiza o peso original usando a Taxa de Aprendizado.

Por que "Back" propagation?



- O erro é calculado **no final** (na saída).
- Para saber a "culpa" de W_1 (lá no começo), precisamos **propagar o erro de volta**.
- Cada camada "repassa" uma parte da culpa para a camada anterior via Regra da Cadeia.
- **Analogia:** é como o chefe culpar o gerente, que culpa o coordenador, que culpa o estagiário — cada um recebe sua parcela de responsabilidade pelo erro!

Metáfora III: Backprop e as Engrenagens do Relógio



Engrenagens Conectadas

- Se o ponteiro final atrasar 5 minutos (erro no final), giramos o mecanismo de trás para frente.
- A Regra da Cadeia multiplica as razões de transmissão locais (derivadas parciais).
- Cada engrenagem (peso) recebe sua parcela exata de ajuste para o relógio funcionar perfeitamente.

Batch, Mini-Batch e SGD

Na prática, não calculamos o gradiente com **todas** as amostras de uma vez.

	Vantagem	**Desvantagem**
Batch GD (todas as N amostras)	Convergência suave	Lento, memória pesada
SGD Puro (1 amostra por vez)	Ultra-rápido	Instável, ruidoso
Mini-Batch (32–128 amostras)	Melhor dos dois mundos	Precisa ajustar `batch_size`

Padrão da Indústria

`model.fit(X, y, epochs=50, batch_size=32)` — Mini-Batch é o default do Keras.

Otimizadores Modernos: Além do SGD

O SGD básico tem limitações (oscila, fica preso). A indústria usa **variações mais inteligentes**:

Otimizador	Ideia Central	Quando Usar
SGD	$W \leftarrow W - \alpha \nabla W$	Baseline simples
SGD + Momentum	Acumula "velocidade" como uma bola descendo a montanha	Superfícies com vales estreitos
Adam	Learning Rate **adaptativo** por peso + momentum	**Default moderno**

Na Prática

`model.compile(optimizer='adam', loss='mse')` — Adam funciona bem "de fábrica" para 90% dos problemas.

Guia Rápido de Diagnóstico

Sintoma	Diagnóstico	Solução
Loss = `NaN`	Exploding Gradient	Reduzir α (ex: 0.001)
Loss não desce	Vanishing Gradient **ou** α minúsculo	Usar ReLU, aumentar α
Loss oscila sem convergir	α grande demais	Reduzir α por fator de 10
Loss converge muito devagar	α pequeno demais	Aumentar α ou usar Adam
Loss cai e depois sobe	Overfitting	Mais dados, regularização

Dica de Ouro

Sempre plote a **Curva de Aprendizado** (Loss $\times$ Épocas). Ela é o "eletrocardiograma" da sua rede — se a curva está saudável, a rede está saudável.

O Verdadeiro Objetivo dos Frameworks (Keras)

Por que não escrevemos Backpropagation à mão no dia a dia?

- **Cálculo de Derivadas:** Possuem *Autograd*. Montam o grafo computacional e derivam equações de milhares de variáveis automaticamente.
- **Gestão de Hardware (GPU):** Transferem tensores para a Memória de Vídeo (VRAM) para usar núcleos paralelos, algo doloroso em C++.
- **Estabilidade Numérica:** Evitam divisão por zero, limites extremos de `log()` e mitigam surgimentos de `NaN`.

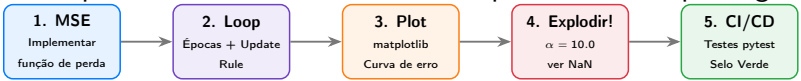
O Papel do Engenheiro de IA

O Keras faz a matemática. Seu trabalho é **físico e arquitetural**: escolher as camadas corretas, a Taxa de Aprendizado e evitar a decoreba dos dados!

Recapitulação — O que Levar Desta Aula

1. A **Função de Perda (MSE)** quantifica numericamente o erro da rede.
2. O **Gradiente** (∇W) aponta para a subida — subtraímos para **descer**.
3. A **Update Rule**: $W_{novo} = W - \alpha \nabla W$ é o coração de todo treinamento.
4. O **Learning Rate** (α) controla o tamanho do passo — grande demais $\rightarrow$ Exploding; pequeno demais $\rightarrow$ Vanishing.
5. O **Backpropagation** usa a Regra da Cadeia para propagar o erro de volta e calcular a "culpa" de cada peso.
6. Na prática, usamos **Adam** (otimizador adaptativo) e **Mini-Batch** (32–128 amostras por vez).

Missão: Implementar o loop de treino, visualizar a descida e provocar um Exploding Gradient de propósito.



Arquivo de Implementação

`src/gradiente.py` — Funções `calcular_mse()`, `treinar()`, `plotar_convergencia()`.

Lab: Parte 2 — A Função MSE

Crie o arquivo `src/gradiente.py` e implemente a MSE:

src/gradiente.py

```
import numpy as np

def calcular_mse(y_real, y_predito):
    return np.mean((y_real - y_predito) ** 2)
```

Lab: Parte 2 — A Função MSE

Crie o arquivo `src/gradiente.py` e implemente a MSE:

src/gradiente.py

```
import numpy as np

def calcular_mse(y_real, y_predito):
    return np.mean((y_real - y_predito) ** 2)
```

Verificação Mental

Se $y_{real} = [3, 5]$ e $y_{pred} = [1, 5]$: $\text{MSE} = \frac{(3-1)^2+(5-5)^2}{2} = \frac{4+0}{2} = 2.0$ ✓

Lab: Parte 3 — O Loop de Treinamento

O coração do aprendizado traduzido em Python:

src/gradiente.py (continuação)

```
def treinar(learning_rate, epochs):
    peso = 10.0           # longe do mínimo
    historico_erros = []

    for epoch in range(epochs):
        erro = peso ** 2              # L(w) = w²
        historico_erros.append(erro)
        gradiente = 2 * peso          # dL/dw = 2w
        peso = peso - (learning_rate * gradiente)

    return historico_erros
```

Lab: Parte 3 — O Loop de Treinamento

O coração do aprendizado traduzido em Python:

src/gradiente.py (continuação)

```
def treinar(learning_rate, epochs):
    peso = 10.0           # longe do mínimo
    historico_erros = []

    for epoch in range(epochs):
        erro = peso ** 2              # L(w) = w²
        historico_erros.append(erro)
        gradiente = 2 * peso          # dL/dw = 2w
        peso = peso - (learning_rate * gradiente)

    return historico_erros
```

Lab: Parte 5 — Provocando o Exploding Gradient

Agora vamos quebrar tudo de propósito!

Resultado com $\alpha = 0.1$

```
>>> treinar(0.1, 5)
[100.0, 64.0, 40.96,
 26.21, 16.78]
```

✓ Erro caindo!

Lab: Parte 5 — Provocando o Exploding Gradient

Agora vamos quebrar tudo de propósito!

Resultado com $\alpha = 0.1$

```
>>> treinar(0.1, 5)
[100.0, 64.0, 40.96,
 26.21, 16.78]
```

✓ Erro caindo!

Resultado com $\alpha = 10.0$

```
>>> treinar(10.0, 5)
[100.0, 36100.0,
 13032100.0, ..., inf]
```

× Erro EXPLODINDO!

Lab: Parte 5 — Provocando o Exploding Gradient

Agora vamos quebrar tudo de propósito!

Resultado com $\alpha = 0.1$

```
>>> treinar(0.1, 5)
[100.0, 64.0, 40.96,
 26.21, 16.78]
```

✓ Erro caindo!

Resultado com $\alpha = 10.0$

```
>>> treinar(10.0, 5)
[100.0, 36100.0,
 13032100.0, ..., inf]
```

× Erro EXPLODINDO!

Lição

Com $\alpha = 10.0$, o peso "quica" de $+10 \to -190 \to +3610 \to$ `inf`.
Na prática: se viu `NaN` no TensorFlow, **reduza o Learning Rate**.

Dúvidas?

Email: alessio@cefetmg.br
Aula: Segunda e Quarta-Feira, 7h as 8:40h (Sala 39)
WhatsApp: Dúvidas rápidas

Lembrete

Confira sua Issue no GitLab!

Próxima aula: Ecossistema Keras

100% da Aula 3 Concluída!