

# Revisão de Machine Learning e Descida de Gradiente

Eduardo Adame

Curso Introdutório de Redes Neurais

13 de agosto de 2025





- **Nome:** Eduardo Adame
- **Formação:**
  - ▶ Mestrando em Matemática Aplicada e Ciência de Dados (FGV EMAp);
  - ▶ Bacharel em Ciência de Dados e Inteligência Artificial (FGV EMAp).
- **Experiência:** Diversos cursos produzidos/ministrados, ex-consultor por 2 anos e meio no Banco Mundial.
- **Pesquisa:** Estatística bayesiana; quantificação de incerteza, modelos não-paramétricos, inferência exata.
- **Contato:** eadamesalles@gmail.com





## Objetivos

- Fornecer **fundamentos sólidos** em redes neurais e aprendizado profundo
- Desenvolver **intuição** sobre funcionamento dos algoritmos
- Preparar estudantes para **projetos reais** de deep learning

## Metodologia

- Aulas teóricas (slides próprios + outros materiais)
- Exercícios práticos em **Jupyter notebooks**
- Implementação de algoritmos **do zero**
- Uso de frameworks modernos (**Keras/TensorFlow**)



## Fundamentos (Semanas 1-8):

1. Revisão ML + Gradiente
2. Introdução às Redes Neurais
3. Retropropagação
4. Keras + Primeiras Redes
5. Regularização + Dropout
6. Introdução às CNNs
7. Revisão + Dúvidas
8. Avaliação 1

## Tópicos Avançados (Semanas 8-16):

9. Transfer Learning
10. Arquiteturas de CNNs
11. Técnicas Avançadas (Data Aug.)
12. Texto + Word Vectors
13. RNNs para Sequências
14. LSTMs + Aplicações
15. Revisão + Dúvidas
16. Avaliação 2



## Horário e Local:

- Quartas-feiras, 15h-18h
- Primeira aula: 13/08/2025
- Última aula: 26/11/2025

## Material:

- Slides (PDF)
- Notebooks interativos
- Notas de aula
- Repositório GitHub

## Pré-requisitos:

- Python
- Álgebra linear
- Cálculo diferencial
- ML básico

## Avaliação:

- Exercícios semanais
- Projeto final

# Por que revisar Machine Learning?



- Redes neurais são uma **extensão** dos métodos de ML tradicionais
- Muitos conceitos fundamentais são compartilhados:
  - ▶ **Função de custo/perda**
  - ▶ **Otimização via gradiente**
  - ▶ Overfitting vs. underfitting
  - ▶ Validação cruzada
- Entender o **gradiente** é crucial para entender redes neurais
- Comparação de performance: NN vs. métodos tradicionais

## Objetivo da Semana 1

Implementar **descida de gradiente** do zero e entender sua dinâmica

# O que é Machine Learning?



Machine Learning permite que computadores aprendam e façam inferências a partir de dados.

## Programação Tradicional:

- Regras explícitas
- Comportamento determinístico
- Limitado a cenários previstos

## Machine Learning:

- Aprende padrões dos dados
- Generaliza para novos casos
- Adapta-se a complexidade



## Aprendizado Supervisionado

- Dados com **respostas conhecidas**
- Objetivo: prever respostas para novos dados
- Exemplos:
  - ▶ Classificação de e-mails (spam/não spam)
  - ▶ Previsão de preços de imóveis
  - ▶ Diagnóstico médico

## Aprendizado Não-Supervisionado

- Dados **sem respostas**
- Objetivo: encontrar estruturas/padrões
- Exemplos:
  - ▶ Segmentação de clientes
  - ▶ Detecção de anomalias
  - ▶ Redução de dimensionalidade





## Regressão

- Saída: valor **contínuo/numérico**
- Exemplos:
  - ▶ Preço de uma casa: R\$ 450.000
  - ▶ Temperatura amanhã: 23.5°C
  - ▶ Vendas do próximo mês: 1.250 unidades
- Métricas: RMSE, MAE,  $R^2$

## Classificação

- Saída: **categoria/classe**
- Exemplos:
  - ▶ E-mail: spam ou não spam
  - ▶ Imagem: gato, cachorro ou pássaro
  - ▶ Transação: fraudulenta ou legítima
- Métricas: Acurácia, Precisão, Recall, F1

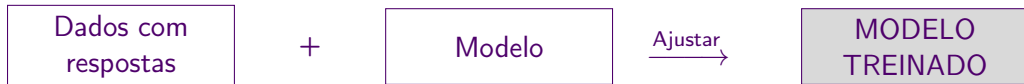


- **Target (Alvo):** Variável que queremos prever
  - ▶ Sinônimos: Resposta, Output, Variável Dependente, Labels
- **Features (Características):** Variáveis usadas para fazer a previsão
  - ▶ Sinônimos: Preditores, Input, Variáveis Independentes, Atributos
- **Example (Exemplo):** Um único ponto de dados
  - ▶ Sinônimos: Observação, Registro, Instância, Linha
- **Label (Rótulo):** Valor do target para um exemplo específico
  - ▶ Sinônimos: Resposta, Valor-y, Categoria

# Pipeline de Aprendizizado Supervisionado



## Fase de Treinamento



## Fase de Predição



## Por que dividir os dados?

- Avaliar **generalização** do modelo
- Detectar **overfitting**
- Simular cenário real de uso

## Divisão típica:

- 70-80% para treino
- 20-30% para teste
- Opcional: conjunto de validação

## Processo

1. **Treinar** com dados de treino
2. **Prever** nos dados de teste
3. **Comparar** previsões com valores reais
4. **Calcular** métricas de erro

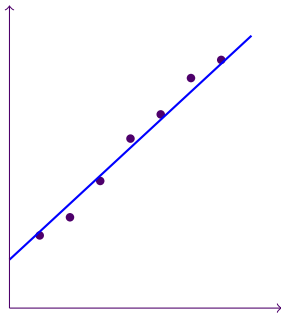
## Importante

NUNCA use dados de teste durante o treinamento!

# Underfitting vs. Overfitting

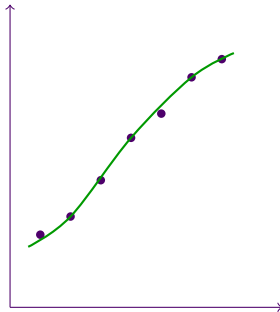


## Underfitting



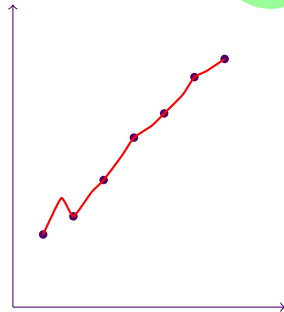
- Modelo muito simples
- Alto bias
- Baixa variância
- Erro alto no treino e teste

## Na medida certa



- Complexidade adequada
- Balanceado
- Boa generalização
- Erro baixo no treino e teste

## Overfitting



- Modelo muito complexo
- Baixo bias
- Alta variância
- Erro baixo no treino, alto no teste



## Para Regressão:

- RMSE (Root Mean Square Error)

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

- MAE (Mean Absolute Error)

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

## Para Classificação:

- Acurácia

$$\text{Acc} = \frac{\text{Previsões Corretas}}{\text{Total de Previsões}}$$

- Precisão e Recall

$$\text{Prec} = \frac{TP}{TP + FP}, \quad \text{Rec} = \frac{TP}{TP + FN}$$

# Problema de Regressão Linear



**Modelo:**  $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$

**Dados conhecidos:**

- $\beta_0 = 1.5$  (intercepto)
- $\beta_1 = 2$  (coef. de  $x_1$ )
- $\beta_2 = 5$  (coef. de  $x_2$ )
- $x_1, x_2 \sim \text{Uniforme}[0, 10]$
- $\varepsilon \sim \mathcal{N}(0, 0.5^2)$

**Objetivo:**

- Estimar  $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$
- Métodos a comparar:
  1. Solução analítica
  2. Scikit-learn
  3. Descida de gradiente

**Função de Custo (MSE)**

$$J(\beta) = \frac{1}{2n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{2n} \sum_{i=1}^n (y_i - \mathbf{x}_i^T \beta)^2 \quad (1)$$

# Solução Analítica vs. Numérica



## Solução Analítica:

$$\hat{\beta} = (X^T X)^{-1} X^T \mathbf{y} \quad (2)$$

### Vantagens:

- Solução exata
- Uma única operação
- Computacionalmente rápida (problemas pequenos)

### Desvantagens:

- Inversão de matriz:  $O(n^3)$
- Problemas numéricos
- Não escala para grandes dados

## Descida de Gradiente:

$$\beta^{(t+1)} = \beta^{(t)} - \alpha \nabla J(\beta^{(t)}) \quad (3)$$

### Vantagens:

- Escala para grandes dados
- Aplicável a funções não-lineares
- Base para redes neurais
- Flexibilidade

### Desvantagens:

- Solução aproximada
- Hiperparâmetros ( $\alpha$ , iterações)
- Convergência pode ser lenta



# Intuição Geométrica do Gradiente

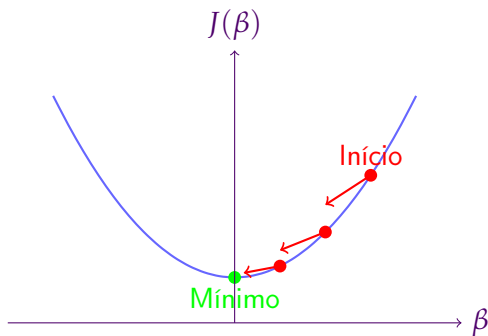


**Imagine:** Você está no topo de uma montanha e quer descer até o vale mais baixo.

## Estratégia:

1. Olhe ao seu redor
2. Encontre a direção mais íngreme
3. Dê um passo nessa direção
4. Repita até chegar ao vale

O **gradiente** aponta para a direção de maior subida. Logo, o **gradiente negativo** aponta para a maior descida!



Trajetória da Descida de Gradiente



## Algoritmo Básico

1. **Inicialização:**  $\beta^{(0)}$  = valores iniciais
2. **Para**  $t = 0, 1, 2, \dots$  **até convergência:**
  - 2.1 Calcular predições:  $\hat{\mathbf{y}} = X\beta^{(t)}$
  - 2.2 Calcular erro:  $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$
  - 2.3 Calcular gradiente:  $\nabla J = -\frac{1}{n}X^T\mathbf{e}$
  - 2.4 Atualizar parâmetros:  $\beta^{(t+1)} = \beta^{(t)} - \alpha\nabla J$
3. **Retornar:**  $\beta^{(T)}$

## Gradiente do MSE:

$$\frac{\partial J}{\partial \beta_j} = -\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i) \cdot x_{ij} \quad (4)$$

# Batch vs. Stochastic Gradient Descent



## Batch Gradient Descent:

- Usa **todos** os dados por iteração
- Gradiente exato
- Convergência suave
- Computacionalmente caro

$$\nabla J = \frac{1}{n} \sum_{i=1}^n \nabla J_i \quad (5)$$

## Stochastic Gradient Descent:

- Usa **uma amostra** por iteração
- Gradiente aproximado
- Convergência com ruído
- Computacionalmente eficiente

$$\nabla J \approx \nabla J_i \quad (6)$$

## Vantagens do SGD

- Escala para milhões de amostras
- Pode escapar de mínimos locais (ruído = benefício!)
- Convergência online (dados chegando continuamente)

# SGD: Ordem dos Dados Importa!



## Sem embaralhamento:

- Mesma ordem a cada época
- Padrões sistemáticos
- Convergência tendenciosa
- Ciclos na trajetória

**Problema:** Se os dados têm ordem específica, o algoritmo pode “memorizar” essa ordem.

## Com embaralhamento:

- Ordem aleatória a cada época
- Reduz viés sistemático
- Convergência mais robusta
- Melhor exploração

**Solução:** Embaralhar dados no início de cada época.

## Exercício no Notebook

Implemente SGD com e sem embaralhamento e compare as trajetórias!



## 1. Comparação de Métodos:

- ▶ Os três métodos convergem para o mesmo resultado?
- ▶ Qual é o mais rápido? Qual é o mais preciso?

## 2. Experimentos com Taxa de Aprendizado:

- ▶  $\alpha = 0.00001$  (muito pequeno)
- ▶  $\alpha = 0.001$  (adequado)
- ▶  $\alpha = 0.1$  (muito grande)

## 3. Visualização da Trajetória:

- ▶ Plotar o caminho no espaço de parâmetros
- ▶ Plotar a evolução da função de custo
- ▶ Identificar ponto de convergência

## 4. SGD com Embaralhamento:

- ▶ Implementar versão estocástica
- ▶ Comparar com e sem embaralhamento

# Critérios de Convergência



Como saber quando parar o algoritmo?

1. Convergência do gradiente:

$$\|\nabla J(\boldsymbol{\beta}^{(t)})\| < \varepsilon_1 \quad (7)$$

2. Convergência dos parâmetros:

$$\|\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\beta}^{(t)}\| < \varepsilon_2 \quad (8)$$

3. Convergência da função de custo:

$$|J(\boldsymbol{\beta}^{(t+1)}) - J(\boldsymbol{\beta}^{(t)})| < \varepsilon_3 \quad (9)$$

4. Número máximo de iterações:  $t > T_{\max}$



**Análise visual é fundamental para entender convergência:**

## Plots importantes:

- Trajetória no espaço de parâmetros
- Evolução da função de custo
- Curvas de nível + trajetória
- Comparação SGD vs. Batch GD

## O que observar:

- Convergência suave vs. oscilante
- Velocidade de convergência
- Ponto final vs. solução analítica
- Efeito da taxa de aprendizado



## Limitações do GD básico:

- Pode ficar **oscilando** em vales estreitos
- Convergência **lenta** em direções de pouca curvatura
- Dificuldade com **mínimos locais**

### Definition 1: SGD com Momentum

$$\mathbf{v}^{(t+1)} = \gamma \mathbf{v}^{(t)} + \alpha \nabla J(\boldsymbol{\beta}^{(t)}) \quad (10)$$

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \mathbf{v}^{(t+1)} \quad (11)$$

onde  $\gamma \in [0, 1)$  é o coeficiente de momentum.

**Algoritmos mais modernos:** Adam, RMSprop, AdaGrad (veremos nas próximas aulas)





## O que aprendemos hoje

- Revisão de conceitos fundamentais de ML
- Regressão linear como base para redes neurais
- Descida de gradiente: algoritmo e implementação
- Importância da taxa de aprendizado
- SGD vs. Batch GD
- Efeito do embaralhamento em SGD
- Critérios de convergência e visualização
- Introdução ao momentum



## Próxima semana: Introdução às Redes Neurais

- Do neurônio biológico ao artificial
- **Perceptron** e funções de ativação
- Primeira rede neural **multi-camadas**
- Por que precisamos de **não-linearidade**?

Obrigado!

Alguma dúvida?

Agora vamos para os exercícios!