Inteligência Artificial 2

Overfitting, Multiclasse e Softmax

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, migramos do NumPy artesanal para o **ecossistema Keras**:

- Ingerimos dados tabulares com `pandas` e normalizamos com `StandardScaler`.
- Construímos um modelo `Sequential` com `Dense` + ReLU + Sigmoid.
- Treinamos com `compile()` + `fit()` e atingimos acurácia $> 65\%$.

A Pergunta de Hoje

O modelo atingiu 95% de acurácia no treino.
Isso significa que ele **aprendeu**... ou **decorou**?

Objetivos da Sessão

- **Overfitting:** Entender por que acurácia alta no treino pode ser uma armadilha.
- **Viés vs Variância:** O dilema fundamental do Machine Learning.
- **A Tríade:** Dividir dados em Treino, Validação e Teste.
- **Diagnóstico Visual:** Plotar as curvas `loss` vs `val_loss`.
- **Softmax:** Migrar de binário para classificação multiclasse.
- **Categorical Crossentropy:** A loss correta para K classes.

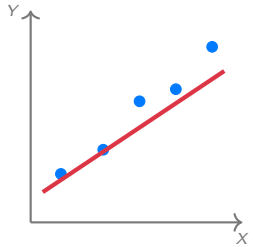
A Analogia do Aluno Decorador

- Um aluno recebe 100 exercícios resolvidos.
- Em vez de entender *por que* funciona, decora: "A questão 4 dá 42".
- Prova **idêntica** à lista → nota 10.
- Professor muda uma vírgula → **nota zero**.

Overfitting = Decoreba

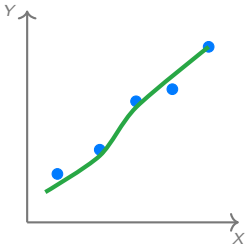
A rede tem memória para decorar **cada exemplo**.
Acurácia 99% no treino + fracasso em dados novos = modelo **inútil**.

A Tríade do Ajuste: Under / Bom / Over



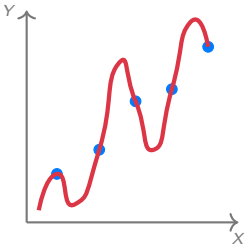
Underfitting

Alto Viés



Bom Ajuste

Generaliza bem



Overfitting

Alta Variância

Exemplo do Mundo Real: Under vs Over

Underfitting: O Simplório

Um corretor de imóveis que diz: "toda casa custa R$ 200 mil, independente do tamanho".

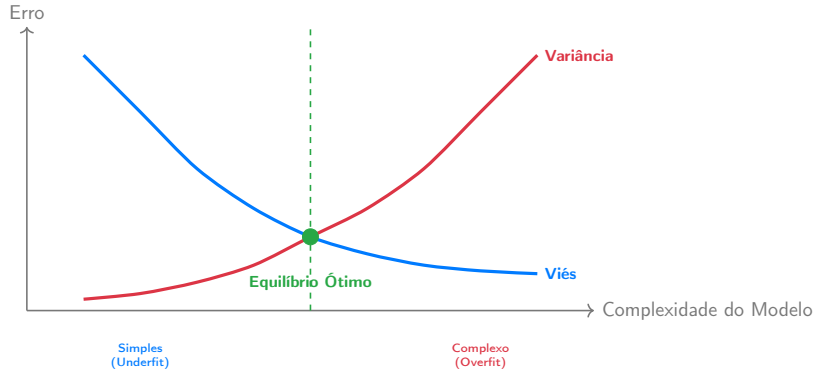
- Erra nas casas pequenas.
- Erra nas mansões.
- Modelo muito simples (rígido).

Overfitting: O Paranóico

O corretor diz: "A casa custa R$ 312.450 porque é amarela, tem um arranhão na porta esquerda e foi vista numa terça-feira".

- Preço exato para *aquela* casa.
- Se pintar a porta, ele chuta errado.
- Decorou ruído, não o padrão.

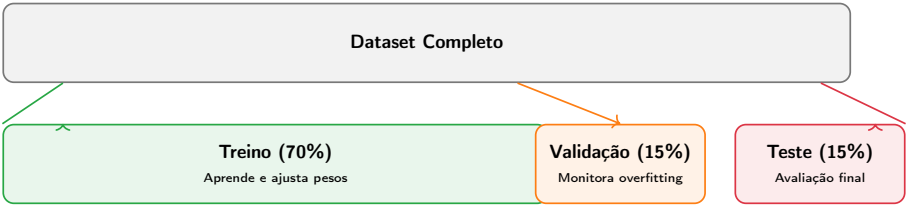
Viés vs Variância: O Dilema Fundamental



- **Viés alto**: modelo simples demais → não aprende os padrões.
- **Variância alta**: modelo complexo demais → decora o ruído.
- **O segredo**: encontrar a complexidade que **minimiza ambos**.

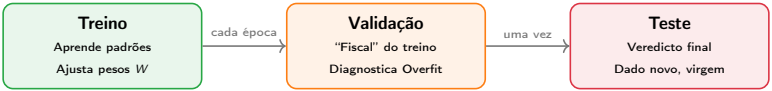
Quando o Overfitting Acontece?

A Vacina: Dividir os Dados em 3 Blocos



- **Treino:** a rede aprende e ajusta W.
- **Validação:** "prova surpresa" a cada época (sem ajustar pesos).
- **Teste:** "cofre" aberto **uma única vez** no final.

Por que 3 Blocos e Não 2?



O Perigo de Só 2 Blocos

Se otimizarmos hiperparâmetros (epochs, camadas, etc.) olhando o **teste**, estamos "vazando" informação do teste para o modelo. O teste deve ser visto **uma única vez**, como um exame final.

O Perigo do Data Leakage (Vazamento)

O que acontece se a rede "ver"o Teste durante o treino?

- **Data Leakage** é quando informações de fora do conjunto de treino vazam para dentro dele.
- Exemplo: Aplicar o `StandardScaler.fit()` no dataset **inteiro** antes de dividir. A média geral agora embute o conhecimento do teste!
- Quando a rede for avaliada no Teste, ela tirará 10, pois já sabia sutilmente como ele era.

Consequência

Na vida real (produção), quando dados 100% inéditos chegarem, o modelo vai fracassar vergonhosamente, gerando prejuízos financeiros severos.

Keras: validation_split

O Keras simplifica a separação de validação:

```
historico = model.fit(
    X_train, y_train,
    epochs=200,
    validation_split=0.2,  # 20% do treino vira validacao
    verbose=0
)
```

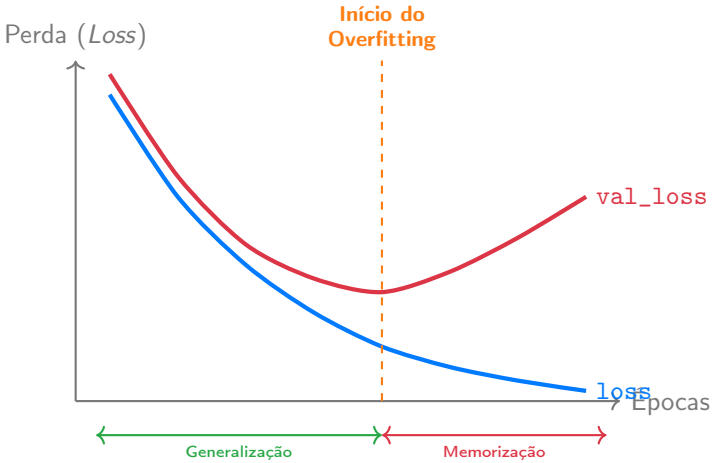
Agora o log exibe **duas métricas** por época:

- `loss` — erro no **treino**.
- `val_loss` — erro na **validação** (sem retropropagar!).

Resultado

`Epoch 50:  loss=0.12 - val_loss=0.45` → Overfitting!

A Assinatura Visual do Overfitting



Quando `val_loss` sobe e `loss` desce: a rede está **decorando**.

Lendo o Gráfico na Prática

Cenário	loss	val_loss	Diagnóstico
Underfitting	Alto e estável	Alto e estável	Rede simples demais
Bom Ajuste	Cai gradualmente	Cai junto	**Continuar!**
Overfit Leve	Continua caindo	Para de cair	Cautela
Overfit Grave	Vai a zero	**Sobe!**	**Parar agora!**

Regra de Ouro

Enquanto as curvas caminham **juntas e caindo**, o modelo está saudável. Quando divergem, o Overfitting começou.

O "Gap"de Generalização

A diferença vertical entre as duas curvas é chamada de **Gap de Generalização**.

- **Gap pequeno:** O modelo se sai no teste tão bem quanto no treino (modelo robusto).
- **Gap grande:** O modelo é um mestre no treino, mas falha no teste (Overfitting).
- **Gap negativo?** (`val_loss` menor que `loss`). Acontece se a regularização (Dropout) é aplicada no treino mas desligada na validação, ou se o lote de validação for "fácil"por sorte.

Plotando no Python

```python
import matplotlib.pyplot as plt

plt.plot(historico.history['loss'], label='Treino')
plt.plot(historico.history['val_loss'], label='Validação')
plt.xlabel('Épocas')
plt.ylabel('Loss')
plt.legend()
plt.title('Diagnóstico de Overfitting')
plt.savefig('overfitting.png')
```

O Que Procurar

- As curvas caminham juntas? → **Saudável.**
- Curva vermelha sobe? → **Overfitting!**

Mudando de Marcha: Além do Sim/Não

Já sabemos avaliar se a rede decorou ou aprendeu.
Agora, vamos ensinar a rede a prever **múltiplas categorias**,
e não apenas respostas binárias.

De Binário para Multiclasse: Softmax

Até agora: Sigmoid $\rightarrow$ saída entre 0 e 1 (binário).

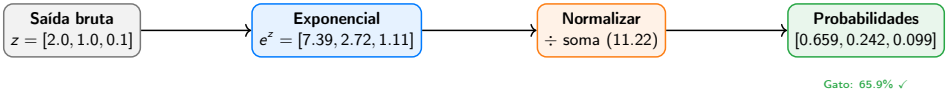
E se o problema tiver K classes? (gato, cachorro, pássaro...)

Função Softmax

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- Converte K valores brutos em **probabilidades** (cada uma entre 0 e 1).
- A **soma** de todas as saídas é exatamente 1.
- A classe predita = aquela com **maior probabilidade**.

Softmax: Pipeline Visual



1. **Exponenciar**: e^{z_i} garante que tudo fique > 0.
2. **Dividir pela soma**: normaliza para somar 1 (distribuição de probabilidade).
3. **Argmax**: a classe com maior valor é a predição.

O Porquê do "Soft"em Softmax

Por que não usar apenas **Max** normal (Hard Max)?

- Hard Max pegaria $[2.0, 1.0, 0.1]$ e transformaria em $[1, 0, 0]$.
- O problema? O Hard Max não é derivável (não tem gradiente liso). E Redes Neurais **dependem de derivadas** para treinar (Backpropagation).
- O **Soft**max suaviza o máximo: a classe dominante ganha a maior probabilidade (ex: 65%), mas as outras classes não zeram completamente.
- Isso cria um "terreno matemático"contínuo, permitindo que a descida do gradiente role suavemente morro abaixo.

Softmax: Exemplo Numérico

Rede produz saídas brutas $z = [2.0,\ 1.0,\ 0.1]$ para 3 classes:

Classe	Cálculo	Probabilidade
Gato	$\frac{e^{2.0}}{e^{2.0}+e^{1.0}+e^{0.1}} = \frac{7.39}{11.22}$	**65.9%**
Cachorro	$\frac{e^{1.0}}{11.22} = \frac{2.72}{11.22}$	24.2%
Pássaro	$\frac{e^{0.1}}{11.22} = \frac{1.11}{11.22}$	9.9%
	Soma	**100%**

A rede escolhe **Gato** com 65.9% de confiança.

Sigmoid vs Softmax: Quando Usar Cada

	Sigmoid	**Softmax**
Uso	Classificação Binária	Classificação Multiclasse
Neurônios Saída	1 neurônio (0 a 1)	K neurônios (soma = 1)
Loss	`binary_crossentropy`	`categorical_crossentropy`
Exemplo	Churn (sim/não)	Dígito (0–9)
Fórmula	$\sigma(z) = \frac{1}{1+e^{-z}}$	$\frac{e^{z_i}}{\sum e^{z_j}}$

Categorical Cross-Entropy: A Loss Multiclasse

A Softmax produz probabilidades, mas como medir o erro? A **Categorical Cross-Entropy** compara a distribuição predita com o rótulo real:

$$\mathcal{L} = -\sum_{i=1}^{K} y_i \cdot \log(\hat{y}_i)$$

Intuição

A loss só penaliza a probabilidade da classe **correta**. Se $\hat{y}_{\text{correta}} \to 1$, $\mathcal{L} \to 0$. Se $\hat{y}_{\text{correta}} \to 0$, $\mathcal{L} \to \infty$.

Multiclasse no Keras

```
# BINÁRIO (Sigmoid)
model_bin = Sequential()
model_bin.add(Input(shape=(10,)))
model_bin.add(Dense(64, activation='relu'))
model_bin.add(Dense(1, activation='sigmoid'))
model_bin.compile(loss='binary_crossentropy', ...)

# MULTICLASSE (Softmax)
model_multi = Sequential()
model_multi.add(Input(shape=(10,)))
model_multi.add(Dense(64, activation='relu'))
model_multi.add(Dense(10, activation='softmax'))  # 10 classes
model_multi.compile(loss='categorical_crossentropy', ...)
```

A única diferença: `sigmoid`→`softmax` e `binary`→`categorical`.

O Tropeço Prático: Formato do Gabarito (Y)

Se a Softmax devolve um vetor com K probabilidades (ex: `[0.1, 0.8, 0.1]`), o gabarito Y **precisa estar compatível**!

Opção 1: One-Hot Encoding

O Keras espera $Y = [0, 1, 0]$

```
loss='categorical_crossentropy'
```

Opção 2: Sparse (Inteiros)

Se seu Y for um inteiro ($Y = 1$ para a 2ª classe), use a versão **sparse**:

```
loss='sparse_categorical_crossentropy'
```

Dica de Ouro

Na dúvida, não altere seus dados. Deixe o Y como uma coluna de inteiros (0, 1, 2...) e use `sparse_categorical_crossentropy`. O Keras faz a conversão interna!

Recapitulação — O que Levar Desta Aula

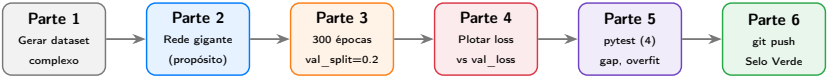
1. **Overfitting** = a rede *decorou* o treino em vez de *aprender* padrões.
2. O dilema **Viés vs Variância**: modelos simples erram por viés, complexos por variância.
3. Dividir dados em **Treino (70%) / Validação (15%) / Teste (15%)** é obrigatório.
4. A **assinatura** do Overfitting: `loss` cai, `val_loss` sobe.
5. Para K classes, usar **Softmax** + `categorical_crossentropy`.
6. A Softmax converte K valores brutos em uma distribuição de probabilidades (soma = 1).

Próxima Aula

Agora que sabemos **diagnosticar** o Overfitting, como **impedi-lo**?
Dropout, **Early Stopping** e **Regularização L2**.

Roteiro do Laboratório 05

Missão: Provocar Overfitting propositalmente e diagnosticá-lo visualmente.

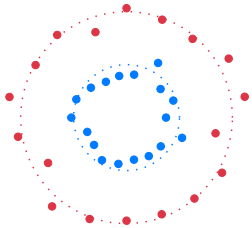


- Rede propositalmente grande (256-128-64) para provocar Overfitting.
- Gráfico `loss` vs `val_loss` é a prova visual do sobreajuste.

Lab Passo 1: O Dataset "Circles"

Vamos usar um dataset sintético do Scikit-Learn (`make_circles`):

- Dados ruidosos, não podem ser separados por uma linha reta.
- Um modelo simples sofreria **Underfitting**.
- Mas nós vamos usar uma rede "monstro"para garantir que ela decore cada gota de ruído, forçando o **Overfitting**.



Lab Passo 2: Construindo a Rede "Monstro"

Como forçamos uma rede a decorar? Dando a ela muitos "neurônios livres":

1. `Dense(256, activation='relu')`
2. `Dense(128, activation='relu')`
3. `Dense(64, activation='relu')`
4. `Dense(1, activation='sigmoid')` (pois é binário).

O Propósito

Esta rede é absurdamente grande para um dataset de 2 features. Ela vai rapidamente decorar a posição exata de cada ponto em vez de aprender a forma geral.

Lab Passo 3: O Treinamento Longo

Não basta ter uma rede gigante, ela precisa de tempo para decorar o gabarito.

- Vocês chamarão o `model.fit()` com um número absurdo de épocas: `epochs=300` ou mais.
- É **obrigatório** passar `validation_split=0.2` (ou usar `validation_data`). Sem isso, o Keras não calcula o `val_loss`.
- Salvem o retorno do fit numa variável: `historico = model.fit(...)`

Lab Passo 4: Diagnosticando Visualmente

A variável `historico` contém um dicionário com tudo o que aconteceu.

```
import matplotlib.pyplot as plt

# Acessando o dicionário history
loss = historico.history['loss']
val_loss = historico.history['val_loss']

# Plotando as duas linhas no mesmo gráfico
plt.plot(loss, label='Loss (Treino)')
plt.plot(val_loss, label='Val Loss (Teste/Val)')
plt.legend()
plt.show()
```

Você verá claramente a curva laranja descolar da azul e voar!

Lab Passo 5: Testes Unitários

Os robôs do Pytest irão avaliar:

- **Teste 1:** Se a arquitetura da rede tem as camadas gigas.
- **Teste 2:** Se a variável `historico` possui chaves de validação.
- **Teste 3 (O Gap):** Verifica se realmente ocorreu overfitting calculando `val_loss - loss`. Se o gap for muito pequeno, você não treinou o suficiente para provocar o problema!

Lab Passo 6: CI/CD Pipeline (GitLab)

Como sempre, o trabalho não acaba até estar versionado:

```
git add .
git commit -m "Laboratorio 05: Overfitting provocado"
git push origin master
```

No GitLab, aguarde a pipeline ficar **Passed**. Se quebrar, leia o erro no terminal, corrija no Jupyter e faça o push de novo.

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: Regularização & Callbacks

100% da Aula 5 Concluída!