Inteligência Artificial 2

Regularização: Dropout e Early Stopping

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

29 de setembro de 2026

Onde Paramos?

Na aula anterior, diagnosticamos o **Overfitting**:

- Uma rede superdimensionada **decora** o treino em vez de aprender padrões.
- Plotamos as curvas `loss` vs `val_loss` e identificamos o marco do sobreajuste.
- Migramos de Sigmoid para **Softmax** (classificação multiclasse).

A Pergunta de Hoje

Sabemos **diagnosticar** o Overfitting.
Agora, como **impedi-lo**?

Objetivos da Sessão

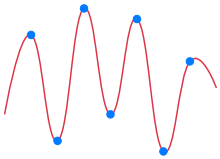
- **Regularização L2:** Penalizar pesos grandes na Função de Perda.
- **Dropout:** Desligar neurônios aleatoriamente durante o treino.
- **Co-adaptação:** Por que neurônios "fofocam" e criam vícios.
- **Early Stopping:** Parar o treino automaticamente antes do Overfitting.
- **Callbacks:** O conceito de vigias automáticos do Keras.
- **Otimizador Adam:** Como o Learning Rate se ajusta sozinho.
- **Receita completa:** A caixa de ferramentas profissional.

Regularização: A Navalha de Ockham

Princípio (séc. XIV)

"Se há múltiplas explicações para um fenômeno, a mais simples é provavelmente a correta."

- Overfitting = a rede escolhe explicações **contorcidas** para decorar.
- **Regularização** = penalizar a complexidade.
- Forçar padrões genéricos, não detalhes do ruído.



Overfit: memoriza tudo

Regularizado: padrão suave

Regularização L2: Penalizando Pesos Grandes

Adicionamos um termo de penalidade à Função de Perda:

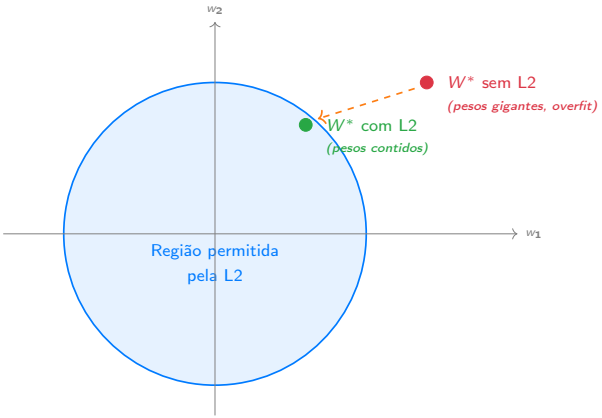
$$\mathcal{L}_{total} = \underbrace{\mathcal{L}_{dados}}_{\text{erro normal}} + \underbrace{\lambda \sum w_i^2}_{\text{penalidade L2}}$$

- Pesos grandes $\rightarrow$ penalidade alta $\rightarrow$ otimizador os reduz.
- Resultado: fronteiras de decisão mais **suaves** e generalizáveis.
- λ controla a intensidade (tipicamente 0.001 a 0.01).

No Keras

```
Dense(64, activation='relu', kernel_regularizer=l2(0.01))
```

Intuição Geométrica da L2



L2 restringe os pesos a uma "esfera" em torno da origem. O ótimo migra de pesos gigantes para pesos **contidos e generalizáveis**.

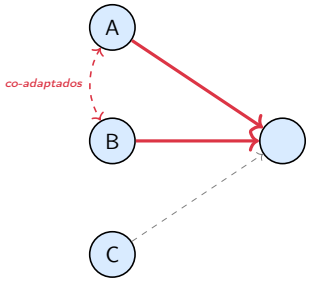
L1 vs L2: Quando Usar Cada Penalidade

	L1 (Lasso)	**L2 (Ridge)**
Penalidade	$\lambda \sum \lvert w_i \rvert$	$\lambda \sum w_i^2$
Efeito	Zera pesos irrelevantes	Reduz pesos (não zera)
Resultado	Modelo **esparso**	Modelo com pesos **difusos**
Ideal para	Seleção de features	Deep Learning em geral
No Keras	`kernel_regularizer=l1()`	`kernel_regularizer=l2()`

Regra Prática

Em Deep Learning, **L2 é uma escolha comum**. L1 pode ser usada quando se deseja induzir esparsidade, mas a escolha deve ser validada.

O Problema da Co-adaptação

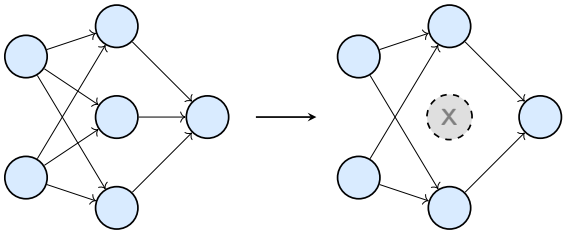


A e B "combinam" para memorizar.
C fica inútil — peso desprezível.

O Vício

Sem Dropout, neurônios A e B criam uma "panelinha" que decora padrões específicos. Neurônio C vira parasita. A rede **perde diversidade** e overfita.

Dropout: Quebrando a Panelinha



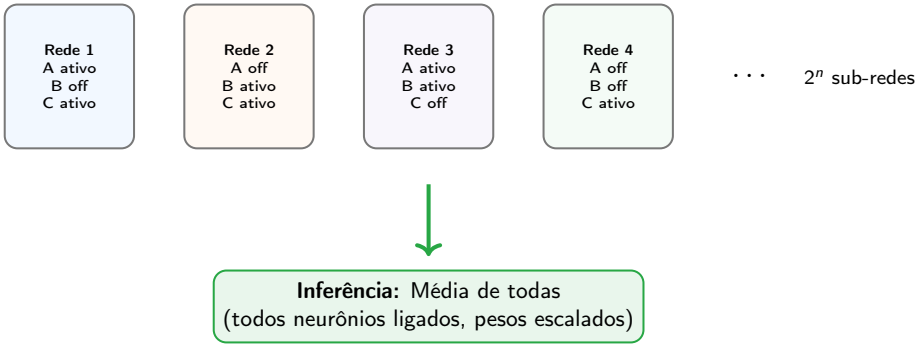
- A cada época, uma % aleatória de neurônios é **desligada**.
- Impede **co-adaptação**: cada neurônio aprende sozinho.
- Cria um *ensemble* implícito de 2^n sub-redes.
- **Só no treino!** Na inferência, todos ligados.

Por que o Dropout Funciona Tão Bem?

A intuição biológica: o aprendizado humano e animal.

- Quando aprendemos a dirigir, no início ficamos sobrecarregados prestando atenção em tudo (co-adaptação extrema).
- Com a experiência, "desligamos"partes da atenção para focar no essencial: não precisamos olhar pro pé para saber onde está o freio.
- O Dropout força a rede a fazer a mesma coisa: como nenhum neurônio pode confiar que seus vizinhos estarão lá na próxima iteração, ele é forçado a **aprender features úteis e independentes** por conta própria.

Dropout: O Ensemble Implícito



Na inferência, ativar todos = calcular a **média** das predições de 2^n redes distintas. É um *ensemble* sem custo extra de treino!

Dropout e o Teste: O Fator de Escala

- Se desligarmos 50% dos neurônios no treino, os pesos dos 50% restantes ficam **maiores** para compensar.
- No **teste**, ligamos TODOS os neurônios. Se deixarmos os pesos gigantes, a saída explodirá!
- Solução: O Keras (TensorFlow) escala automaticamente os pesos no teste (ou escala durante o treino).
- **Vocês não precisam fazer nada, a biblioteca cuida disso sozinha.** Mas é vital saber que a rede se comporta **diferente** no `fit()` e no `evaluate()`.

Dropout no Keras

```
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(10,)))
model.add(Dropout(0.3))   # Desliga 30% dos neurônios
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
```

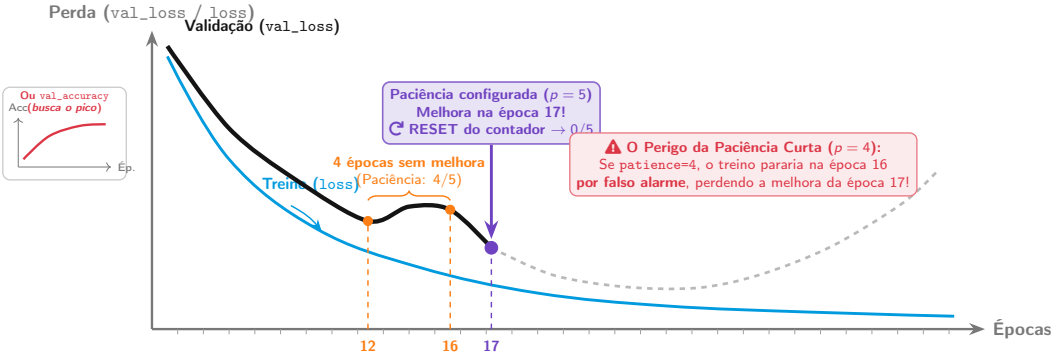
Regra Prática

Taxas entre **0.2 e 0.5**.
Acima de 0.5: risco de Underfitting.

Estratégia

Mais agressivo no início (0.3).
Menos perto da saída (0.2).

Early Stopping: 1. Paciência (*Patience*) e o Reset



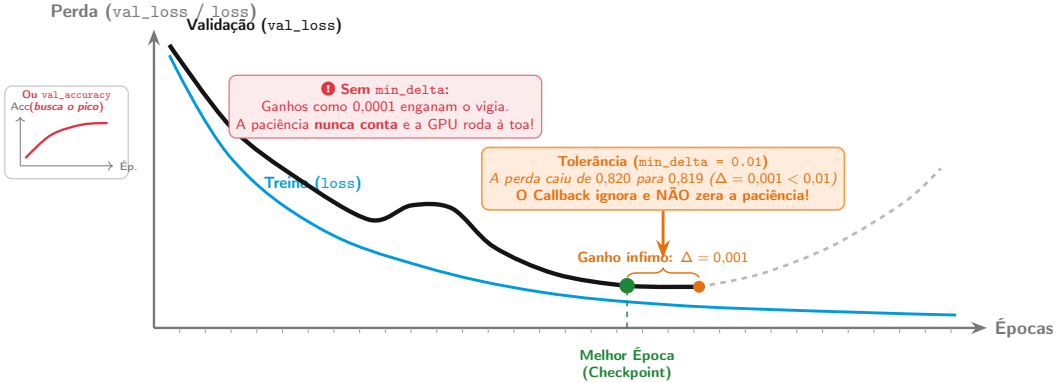
O Dilema da Paciência

Muito curta ($p = 2, 4$): Para precipitadamente diante de qualquer ruído temporário.
Muito longa ($p = 200$): Gasta tempo e energia à toa.

O Mecanismo de Reset

A cada época que atinge um **novo recorde histórico**, o contador de paciência é **zerado imediatamente**, dando novo fôlego à rede.

Early Stopping: 2. Tolerância (min_delta) e Falso Progresso



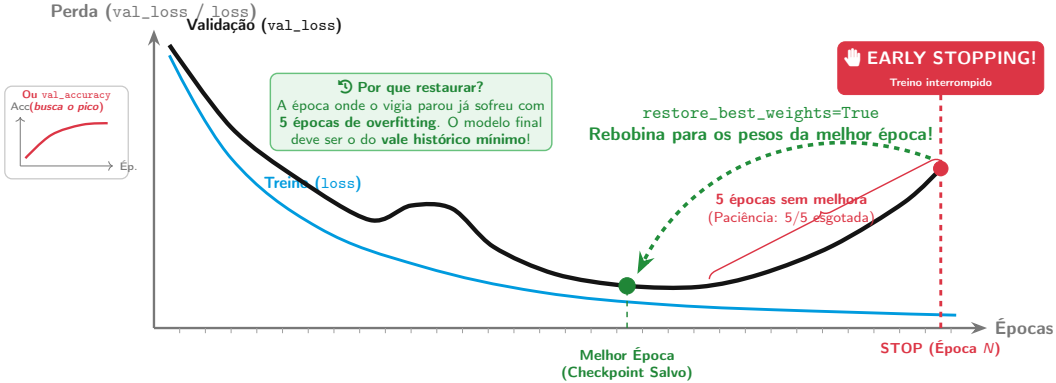
O Problema da Falsa Melhora

No fundo do vale, a loss pode continuar caindo em passos microscópicos (0,001, 0,0001). Sem um limite, o contador nunca dispara.

O Papel do min_delta

`min_delta=0.01` exige um progresso **significativo**. Se o ganho for menor que o delta, a paciência continua correndo rumo à parada.

Early Stopping: 3. Disparo e Rebobinamento de Pesos



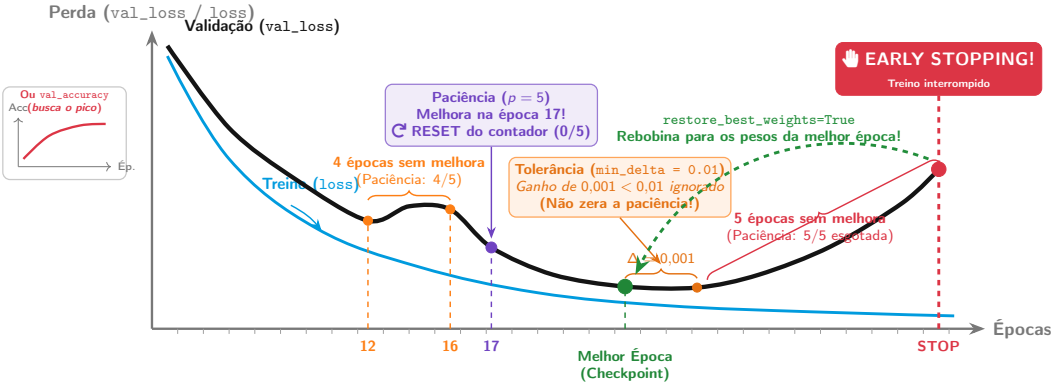
O Ponto de Parada

O Callback só descobre que o vale passou após `patience` épocas subindo. Portanto, o estado final do treino é **subótimo**.

`restore_best_weights=True`

Restaura em memória os pesos do **checkpoint associado à melhor métrica monitorada**; isso não certifica a generalização.

Early Stopping: Anatomia Completa da Decisão



1. Paciência & Reset

Se a perda melhorar antes de esgotar `patience` (ex: ép. 17), o contador volta a zero.

2. Tolerância (`min_delta`)

Ganhos microscópicos ($< 0{,}01$) são ignorados, evitando épocas inúteis.

3. `restore_best_weights`

O vigia interrompe na época com overfit, mas restaura o modelo do vale global.

Como o Early Stopping Decide?

O Parâmetro Crucial

`restore_best_weights=True` restaura os pesos da época $t+1$ (o vale), não os da época $t+6$ (já contaminados por Overfitting).

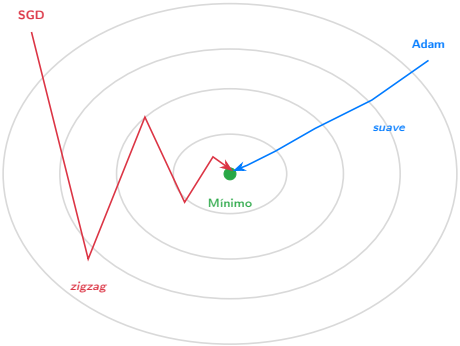
Early Stopping no Keras

```
from tensorflow.keras.callbacks import EarlyStopping

callback = EarlyStopping(
    monitor='val_loss',          # O que vigiar
    patience=10,                 # Tolerância de 10 épocas
    restore_best_weights=True    # ESSENCIAL!
)

model.fit(X_train, y_train,
          epochs=500,             # Teto alto (callback decide)
          validation_split=0.2,
          callbacks=[callback])
```

Por que epochs=500?

Com Early Stopping ativo, o número alto atua como teto. O Callback pode parar antes conforme a métrica, `patience` e `min_delta`; esses parâmetros precisam de validação.

SGD vs Adam: A Descida na Montanha



- **SGD**: Taxa fixa $\rightarrow$ oscila brutalmente em terrenos irregulares.
- **Adam**: Adapta o passo $\rightarrow$ convergência suave e estável.

Otimizadores: Comparativo

	SGD	**Adam**
Learning Rate	Fixo (α manual)	Adaptativo (por peso)
Momentum	Não tem	1º Momento (m)
Curvatura	Ignora	2º Momento (v)
Configuração	Requer tuning manual	"Out of the box"
Convergência	Lenta e ruidosa	Rápida e estável
Uso	Pesquisa avançada	**90% da indústria**

$$W_{novo} = W_{atual} - \alpha \cdot \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

Os Hiperparâmetros do Adam

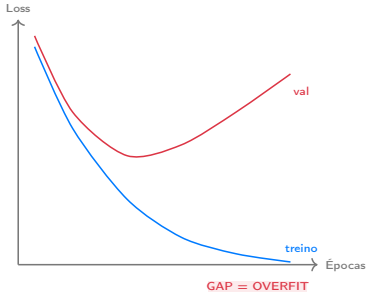
Apesar de ser "Out of the box", o Adam possui botões:

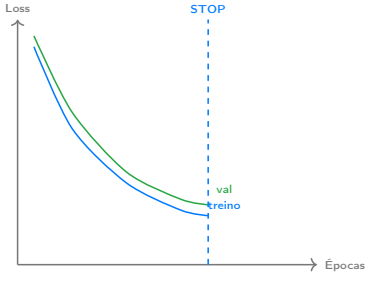
- `learning_rate` (Padrão 0.001): O principal. Se a loss ficar pulando feito pipoca, reduza. Se travar logo cedo, aumente.
- `beta_1` (0.9): Momentum para a média do gradiente.
- `beta_2` (0.999): Momentum para a variância do gradiente.

Dica de Ouro

99% do tempo, você **só altera o learning rate** e deixa os betas quietos.

Antes vs Depois: O Gráfico que Prova





- **Esquerda**: Sem proteção, curvas divergem → gap crescente = Overfitting.
- **Direita**: Com Dropout + Early Stopping, curvas **caminham juntas**.

A Caixa de Ferramentas Completa

Combinando todas: Normalização + L2 + Dropout + Early Stopping = estratégia que pode reduzir sobreajuste e requer validação.

Receita Resumida: O Modelo Profissional

#	Passo	Técnica
1	Separar dados em Treino/Val/Teste	`train_test_split`
2	Normalizar features	`StandardScaler`
3	Construir rede com Dropout entre camadas	`Dropout(0.3)`
4	Compilar com Adam	`optimizer='adam'`
5	Ativar Early Stopping	`EarlyStopping`
6	Avaliar no conjunto de teste	`model.evaluate`

Esta receita é válida para **qualquer** problema de classificação. É a base que usaremos no **TP1**.

Recapitulação — O que Levar Desta Aula

1. **Regularização L2** penaliza pesos grandes: $\mathcal{L}_{total} = \mathcal{L}_{dados} + \lambda \sum w_i^2$.
2. **Dropout** desliga neurônios aleatoriamente, impedindo co-adaptação e criando um ensemble implícito de 2^n sub-redes.
3. **Early Stopping** monitora `val_loss` e para o treino automaticamente com `restore_best_weights`.
4. **Adam** ajusta α individualmente para cada peso (padrão da indústria).
5. A combinação dessas técnicas forma a **caixa de ferramentas profissional** contra Overfitting.

Próxima Aula

Com dados tabulares dominados, é hora do salto: **Redes Convolucionais (CNNs)** e **Transfer Learning** — como IAs enxergam **imagens**.

Roteiro do Laboratório 06

Missão: Curar o Overfitting da Aula 05 com Dropout + Early Stopping.

- Reutiliza o dataset do Lab 05 (comparação justa Antes vs Depois).
- Alvo: gap treino-validação **reduzido**, Early Stopping ativou antes de 500.

Lab Passo 1: O Retorno do "Monstro"

- A ideia da ciência é a reprodutibilidade. Para provar que o "remédio"funciona, precisamos do exato mesmo doente.
- Usaremos a mesma rede gigante do Lab 05: 256, 128, 64 neurônios, e o mesmo dataset Moons.
- Mas agora, injetaremos as "vacinas"entre as camadas densas.

Lab Passo 2: Injetando Dropout

As camadas de Dropout entram **intercaladas** com as Dense.

```
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3)) # Desliga 30% do relu de cima
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3)) # Idem
```

Atenção

Não coloque Dropout antes da camada final (a de saída Sigmoid). O neurônio de resposta final não pode apagar!

Lab Passo 3: O Callback EarlyStopping

Vamos instanciar nosso "robô vigia".

```
from tensorflow.keras.callbacks import EarlyStopping

vigia = EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True
)
```

Isso significa: se o `val_loss` ficar 15 épocas sem atingir um novo mínimo recorde, o vigia interrompe e rebobina a rede para a melhor época da história.

Lab Passo 4: O Treino com Teto Alto

- Com o vigia ativo, você pode (e deve) colocar `epochs=1000`.
- Isso dá margem para o treinamento evoluir, sujeito à métrica e aos parâmetros definidos. O ponto de parada deve ser avaliado no experimento.
- Lembre-se de passar o argumento `callbacks=[vigia]` dentro do método `fit()`. E não esqueça do `validation_split=0.2`!

Lab Passo 5: Testes Unitários ("A Prova")

- **Teste 1:** Verifica se você instanciou Dropout com taxas válidas.
- **Teste 2:** Verifica o Early Stopping (patience, restore_best).
- **Teste 3 (Crucial):** O teste vai acessar a lista de perdas do histórico. Se a época final foi 1000, o vigia falhou. Se a rede parou cedo e o Gap (Treino vs Val) foi **pequeno**, a cura funcionou!

Lab Passo 6: CI/CD Pipeline

O ritual de passagem de sempre:

```
git add .
git commit -m "Laboratorio 06: Dropout e EarlyStopping"
git push origin master
```

Ao receber o Selo Verde, você oficialmente sabe criar redes neurais tabulares com qualidade de **Produção**.

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: Transfer Learning

100% da Aula 6 Concluída!