

REDES NEURAIS

ALGORITMO DE BACKPROPAGATION E
AJUSTE DE HIPERPARÂMETROS

Docente: Dr. Thales Levi Azevedo Valente

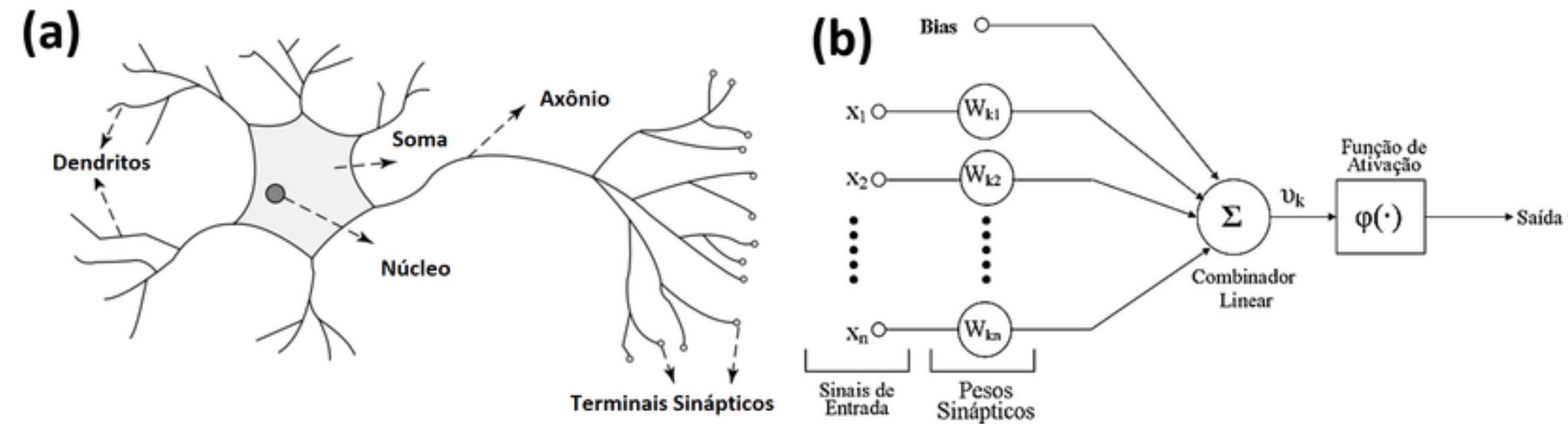
Discentes: Caio Reis, Katarina Ires e Melissa

SUMÁRIO

- 03 O que é uma rede neural
- 04 O que é o Backpropagation
- 05 Importância do Backpropagation
- 07 A estrutura de um MLP
- 10 Analogia com Backpropagation
- 11 Fases do Backpropagation
- 22 Visualização e exemplos no ipynb
- 23 Exemplo prático
- 24 Problemas no Backpropagation
- 33 O que são Hiperparâmetros?
- 35 Principais Hiperparâmetros
- 42 Técnicas para Ajuste de Hiperparâmetros
- 62 Impacto dos Hiperparâmetros
- 66 Técnicas para Evitar Overfitting
- 70 Por que Otimizar Hiperparâmetros é Importante?
- 71 Melhores Práticas para o ajuste de Hiperparâmetros

O QUE É UMA REDE NEURAL?

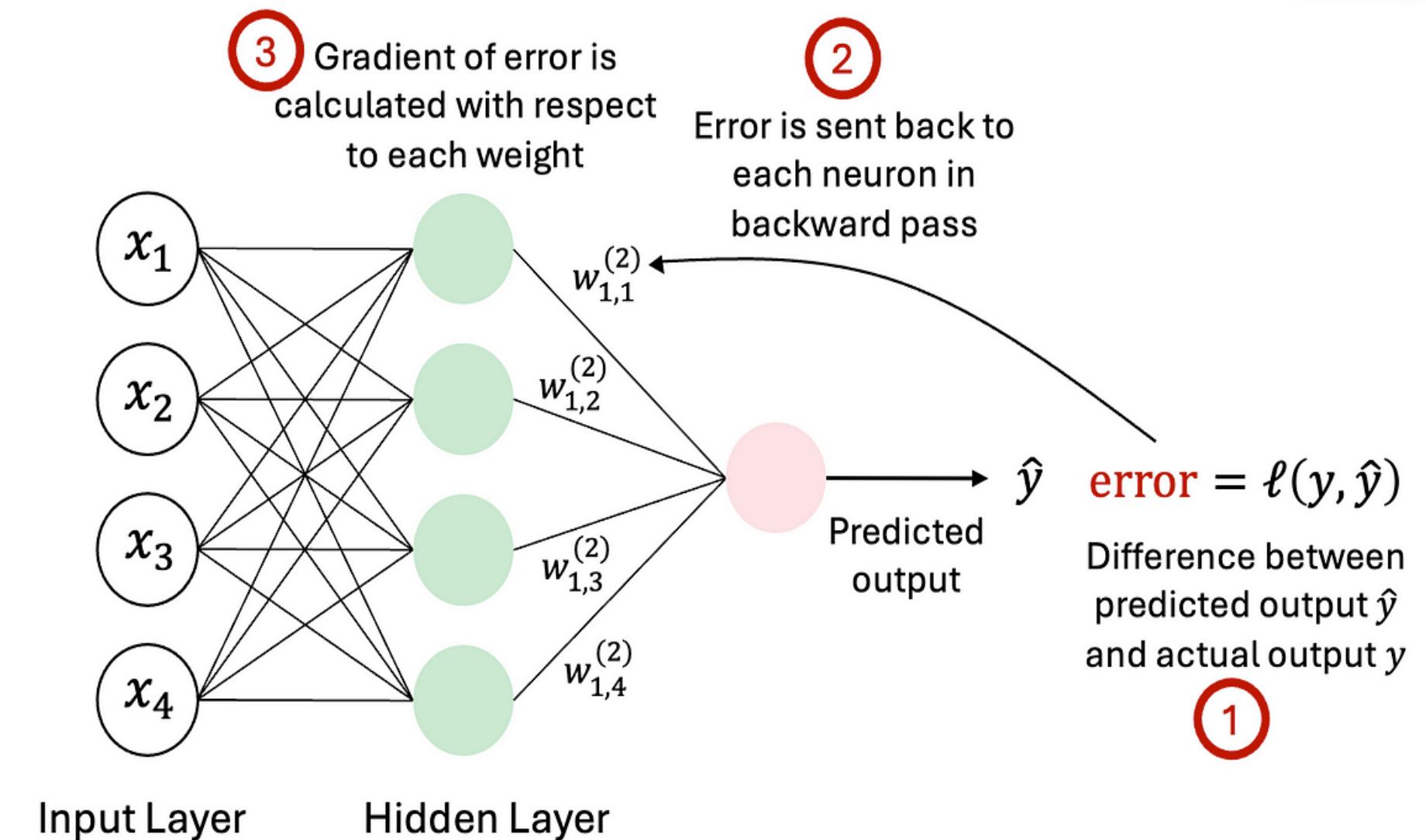
- Inspirada no cérebro humano
- Aprende com dados
- Usa conexões entre “neurônios” artificiais
- Aplica-se em tarefas como: reconhecimento de voz, imagem, texto...



Fonte: Sarmento, Arianne & Dos Santos, Wellington.
(2022)

O QUE É O BACKPROPAGATION?

O Backpropagation (propagação reversa) é um algoritmo fundamental no treinamento de redes neurais artificiais. Desenvolvido na década de 1970 e popularizado em 1986 por Rumelhart, Hinton e Williams, este algoritmo revolucionou o campo do aprendizado de máquina ao fornecer um método eficiente para treinar redes neurais multicamadas

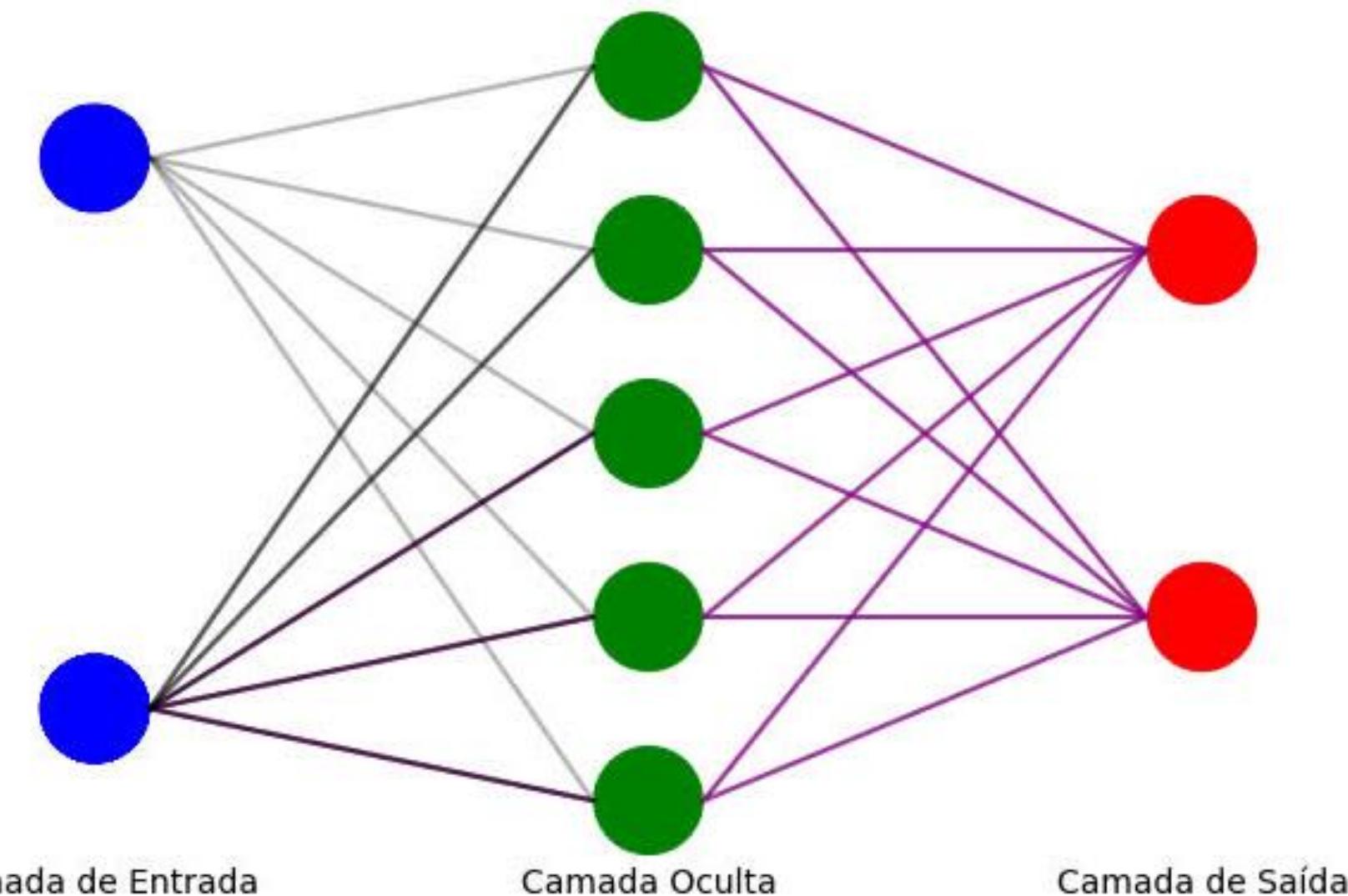


Fonte:
medium

IMPORTÂNCIA DO BACKPROPAGATION

- Viabiliza o treinamento de redes multcamadas (rede com camadas ocultas)
- Aprendizado automático de características
- Base para Deep Learning
- Aplicabilidade universal

Propagação do Erro para Camadas Anteriores

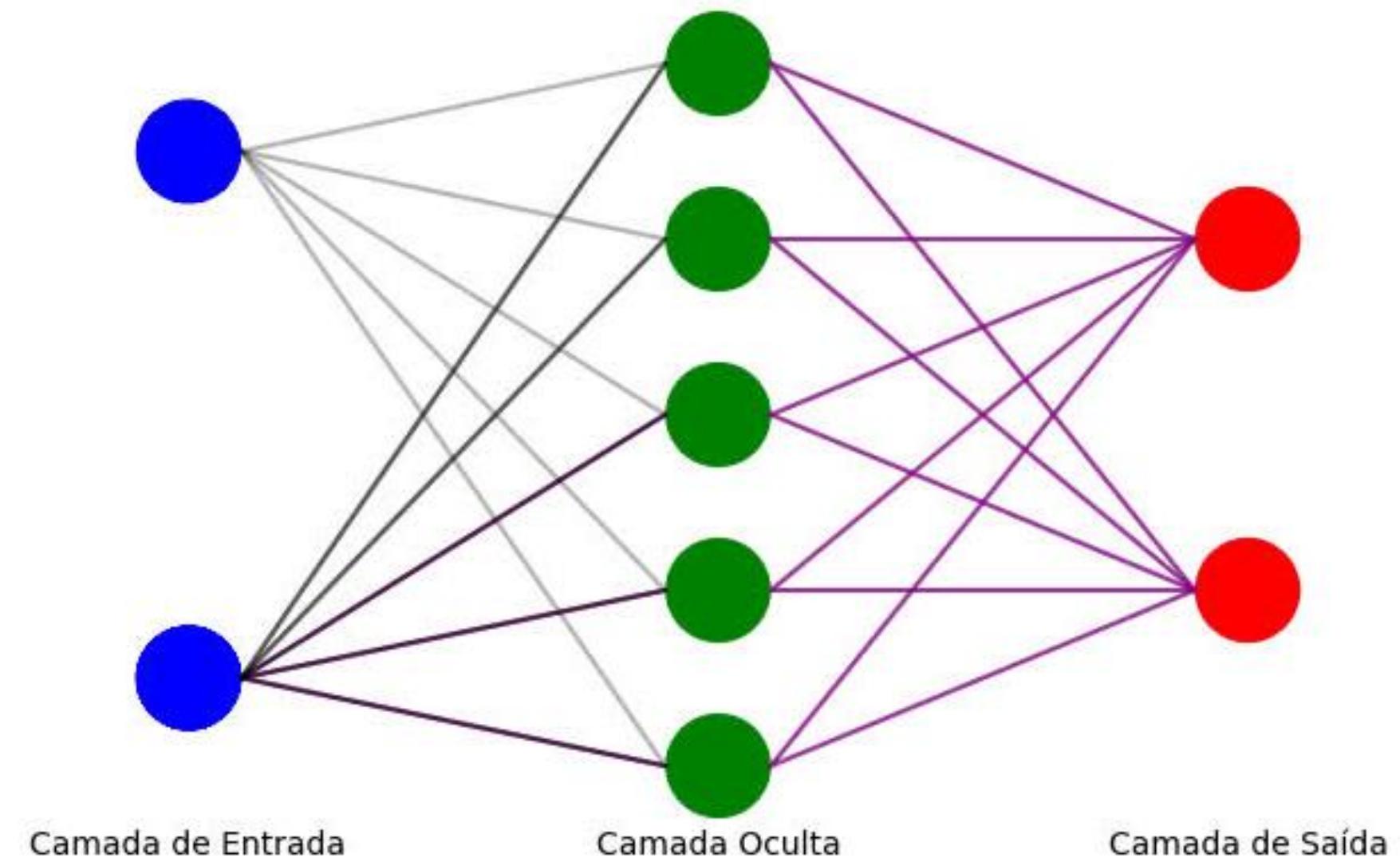


IMPORTÂNCIA DO BACKPROPAGATION

O que são camadas ocultas?

- Viabiliza o treinamento de redes multcamadas (rede com camadas ocultas)
- Aprendizado automático de características
- Base para Deep Learning
- Aplicabilidade universal

Propagação do Erro para Camadas Anteriores



A ESTRUTURA DE UM MLP

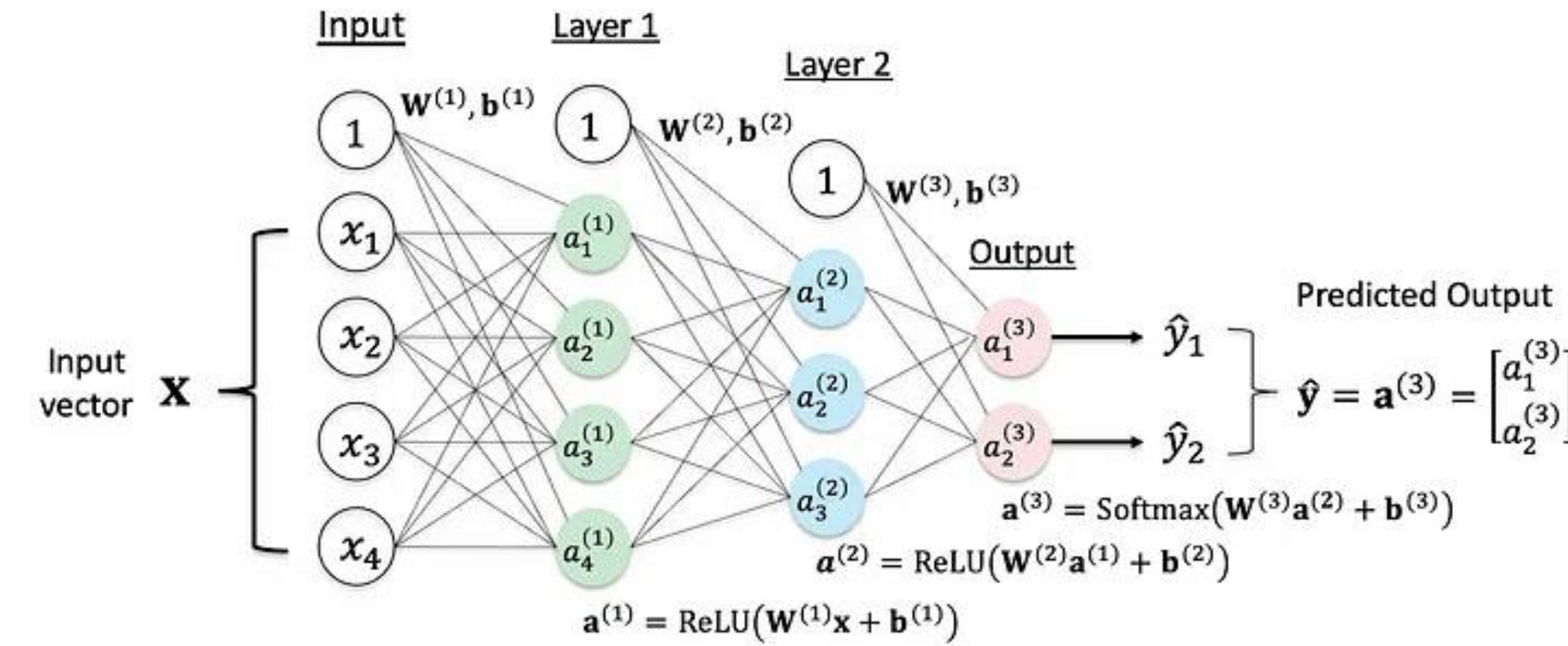
$$\mathbf{a}^{(l)} = g^{(l)}(\mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

$$\mathbf{a}^{(l)} = \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n_l}^{(l)} \end{bmatrix} = g^{(l)} \left(\begin{bmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \dots & w_{1,n_l-1}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \dots & w_{2,n_l-1}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l,1}^{(l)} & w_{n_l,2}^{(l)} & \dots & w_{n_l,n_l-1}^{(l)} \end{bmatrix} \begin{bmatrix} a_1^{(l-1)} \\ a_2^{(l-1)} \\ \vdots \\ a_{n_{l-1}}^{(l-1)} \end{bmatrix} + \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{bmatrix} \right)$$

onde:

- $\mathbf{W}(1)$ e $\mathbf{b}(1)$ são a matriz de peso e o vetor de polarização para a camada 1
- $\mathbf{a}(1)$ é a saída de ativação de todos os neurônios na camada 1
- $g(1)$ é a função de ativação para a camada 1

A ESTRUTURA DE UM MLP



Fonte:
medium

$$\hat{\mathbf{y}} = f_{\theta}(\mathbf{x}) = \text{Softmax}(\mathbf{W}^{(3)} \text{ReLU}(\mathbf{W}^{(2)} \text{ReLU}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)})$$

Fonte:
medium

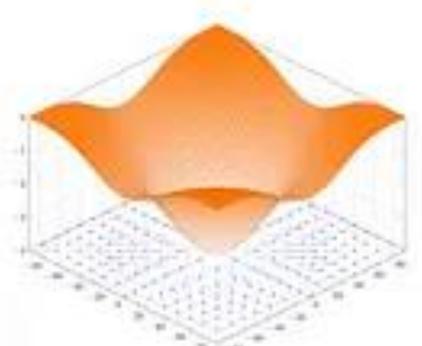
A ESTRUTURA DE UM MLP

- A necessidade de retropropagação:

Uma abordagem ingênua envolveria perturbar cada parâmetro individualmente e observar a variação na função de custo $\mathcal{L}(\theta)$. Esse método é computacionalmente proibitivo, especialmente para redes com milhões de parâmetros

- $\hat{\mathbf{y}} = f_{\theta}(\mathbf{x}) = g^{(L)}(\mathbf{W}^{(L)} \dots g^{(2)}(\mathbf{W}^{(2)}g^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \dots + \mathbf{b}^{(L)})$
- $\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \dots, \mathbf{W}^{(L)}, \mathbf{b}^{(L)}\}$ Parameters of the Neural Network

$$\mathbf{W}^{(l)} = \begin{bmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \dots \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad \mathbf{b}^{(l)} = \begin{bmatrix} b_i^{(l)} \\ \vdots \end{bmatrix}$$



$$\nabla_{\theta} \mathcal{L}(\theta_t) = \begin{bmatrix} \frac{\partial \mathcal{L}(\theta_t)}{\partial w_{i,j}^{(l)}} \\ \vdots \\ \frac{\partial \mathcal{L}(\theta_t)}{\partial b_i^{(l)}} \end{bmatrix}$$

Gradient Descent Algorithm

Initialization: start at θ_0

while ($\theta_{t+1} \neq \theta_t$)

{

compute gradient of θ_t at t and update parameters

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_{\theta} \mathcal{L}(\theta_t)$$

To efficiently compute the gradient when dealing with a large number of parameters, we employ a technique known as backpropagation.

Fonte:
medium

ANALOGIA COM O BACKPROPAGATION?

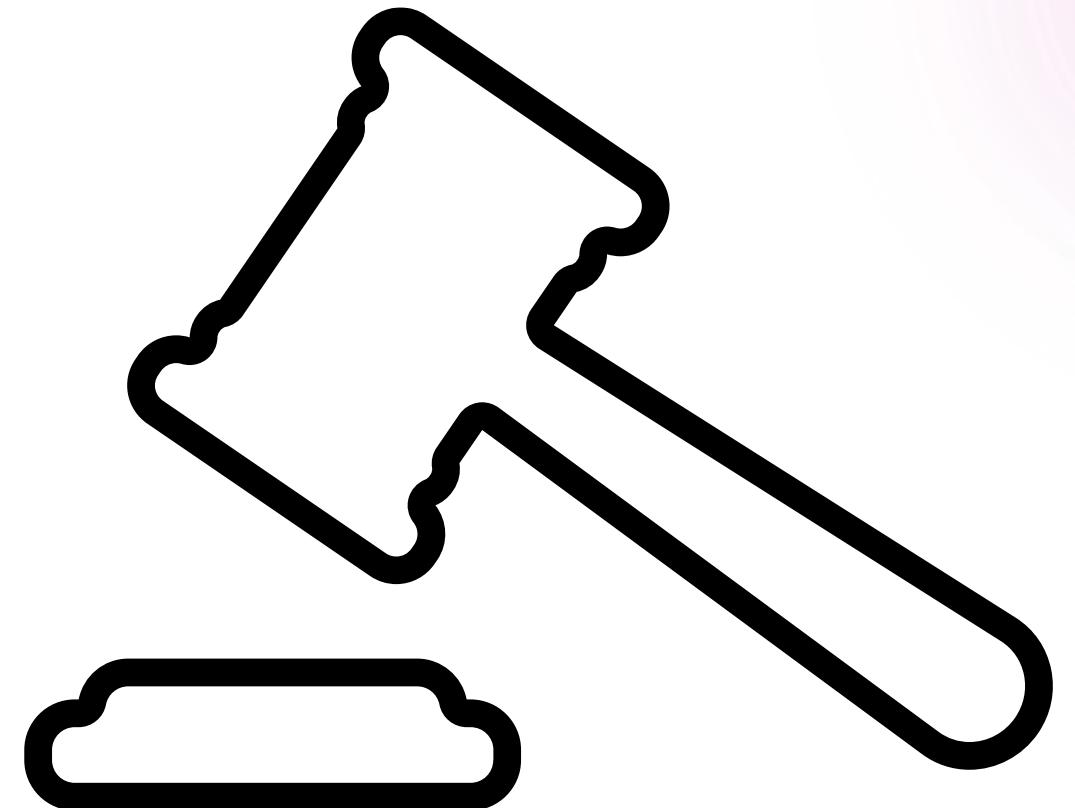
Atribuição de culpa

1. Previsão (Forward Pass)

- A rede recebe os dados de entrada e, camada por camada, calcula uma previsão final

2. Cálculo do Erro

- Compara-se a previsão da rede com o resultado correto para quantificar o "tamanho" do erro



ANALOGIA COM O BACKPROPAGATION?

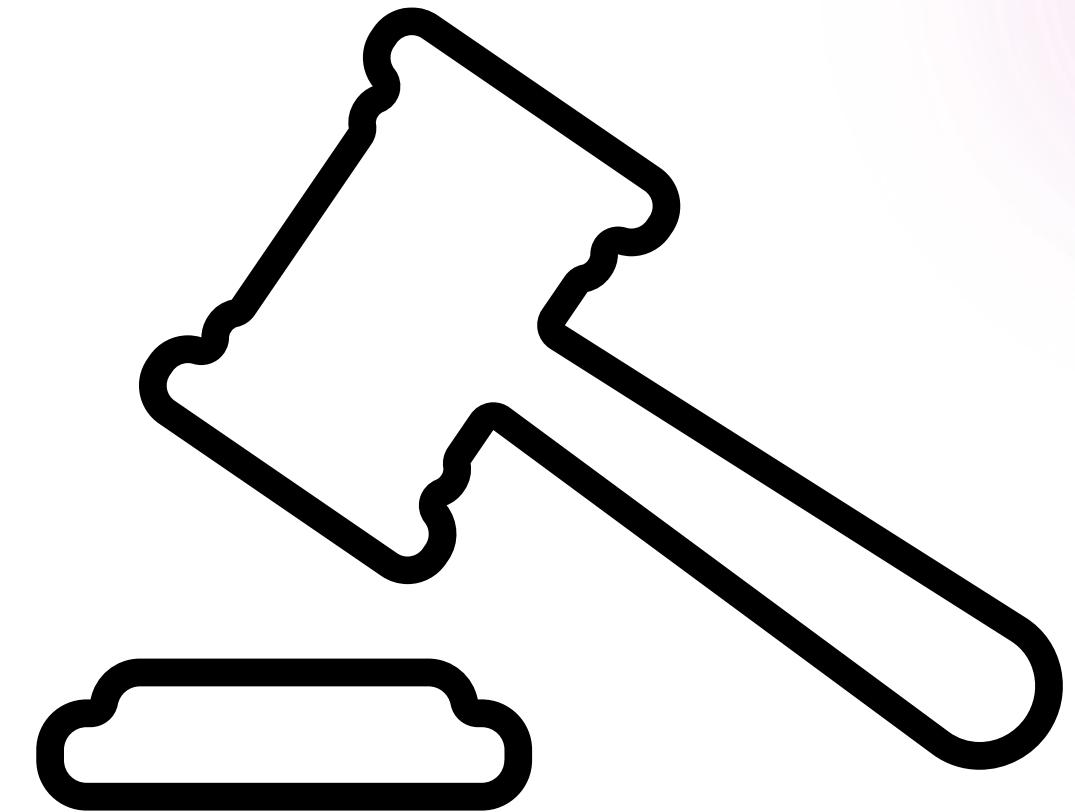
Atribuição de culpa

3. Atribuição de "Culpa" (Backpropagation)

- O erro é propagado de trás para frente
- Para cada peso, calcula-se o gradiente: uma "nota de culpa" que mede sua responsabilidade (direção e magnitude) no erro final
- A "culpa" de um neurônio é baseada na sua influência sobre os neurônios "culpados" da camada seguinte

4. Correção (Otimização com Gradiente Descendente)

- Os pesos são ajustados na direção oposta ao seu gradiente para reduzir o erro
- Pesos com maior "culpa" (maior gradiente) sofrem os maiores ajustes



ANALOGIA COM O BACKPROPAGATION?

Aprender a cozinhar

1. Imagine que cozinhar um risoto é como uma rede neural fazendo uma previsão. A receita é o seu modelo e as quantidades de cada ingrediente são os "pesos".
2. Ao seguir a receita (**forward pass**), você produz o prato final. Ao prová-lo, você nota o **erro**: está salgado demais.



ANALOGIA COM O BACKPROPAGATION?

Aprender a cozinhar

3. Seu cérebro imediatamente faz o **backpropagation**: ele não culpa o prato inteiro, mas rastreia o erro até sua causa específica, atribuindo a "culpa" à quantidade de sal.
4. Para a próxima vez, você faz um ajuste preciso (**gradiente descendente**): usar menos sal. Assim como um cozinheiro refina sua técnica a cada tentativa, a rede neural usa esse ciclo de "provar, culpar e corrigir" para se tornar cada vez mais precisa.



FASES DO BACKPROPAGATION

Propagação do Erro para Camadas Anteriores

01

Feedforward (Propagação Direta)

02

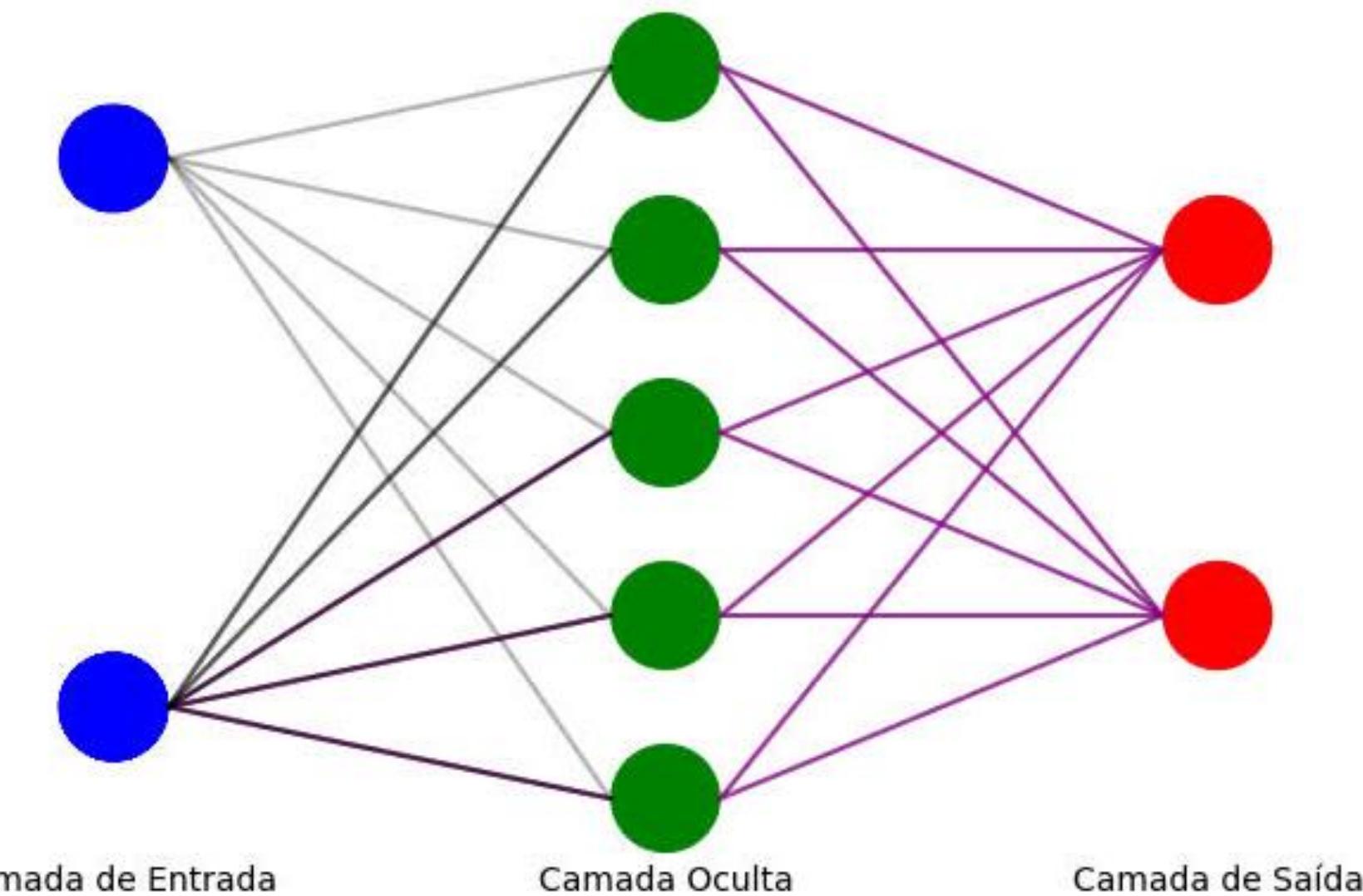
Cálculo do Erro

03

Backward Pass

04

Atualização dos pesos



Feedforward

Matemática do Feedforward

Para cada neurônio em uma camada, realizamos os seguintes cálculos:

$$z_j^l = \sum_{i=1}^n w_{ji}^l a_i^{l-1} + b_j^l$$

Onde:

- z_j^l é a soma ponderada para o neurônio j na camada l
- $w_{jj'}^l$ é o peso da conexão entre o neurônio j' na camada $l - 1$ e o neurônio j na camada l
- $a_{j'}^{l-1}$ é a ativação do neurônio j' na camada $l - 1$
- b_j^l é o viés (bias) do neurônio j na camada l
- n é o número de neurônios na camada $l - 1$

Aplicação da função de ativação:

$$a_j^l = f(z_j^l)$$

Onde:

- a_j^l é a ativação do neurônio j na camada l
- f é a função de ativação (como sigmoid, ReLU, etc.)

Cálculo do Erro

Funções de Erro Comuns

- **Erro Quadrático Médio (MSE - Mean Squared Error)**

O MSE é uma das funções de erro mais comuns para problemas de regressão:

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

Onde:

- E é o erro total
- n é o número de exemplos
- y_i é o valor desejado (target) para o exemplo i
- \hat{y}_i é o valor previsto pela rede para o exemplo i

Suponha que:

- A saída da rede é 0.8
- O valor correto é 1.0
- A função de erro é

$$E = (1.0 - 0.8)^2 = 0.04$$

O backpropagation vai:

Calcular como esse erro depende de cada peso

Atualizar os pesos para reduzir esse erro na próxima rodada de treino

Cálculo do Erro

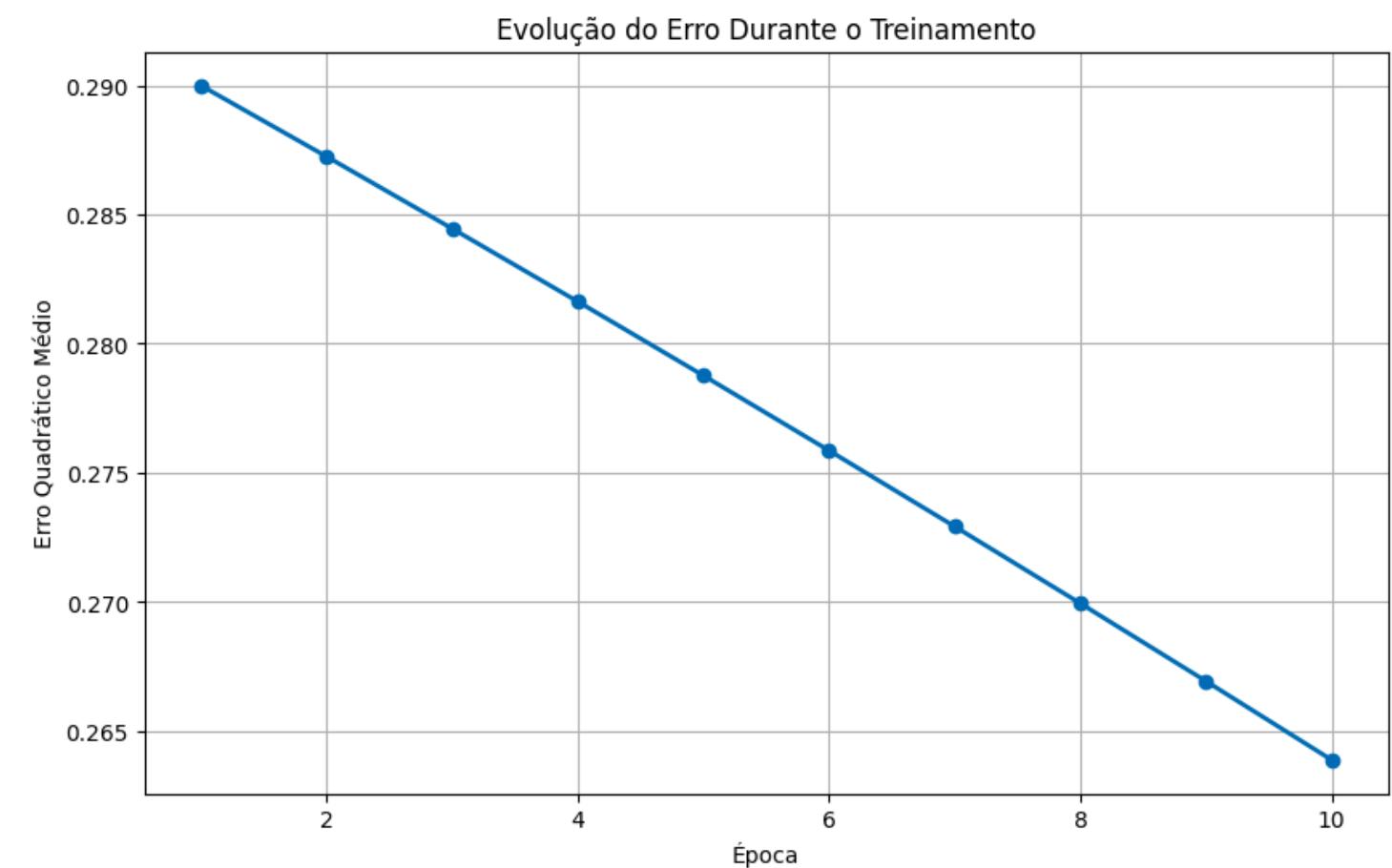
- **Entropia Cruzada (Cross-Entropy)**

A entropia cruzada é frequentemente usada para problemas de classificação:

$$J = -\frac{1}{M} \sum_{j=1}^M \sum_{i=1}^n y_{ij} \log(\hat{y}_{ij})$$

Onde:

- M é o número total de exemplos no seu lote/dataset.
- O primeiro somatório ($j=1$ a M) percorre todos os exemplos.
- O segundo somatório ($i=1$ a n) é a fórmula que você mostrou, calculada para cada exemplo j .



Importância do Cálculo do Erro

- Medida de desempenho



- Direção do aprendizado



- Base para o Backpropagation

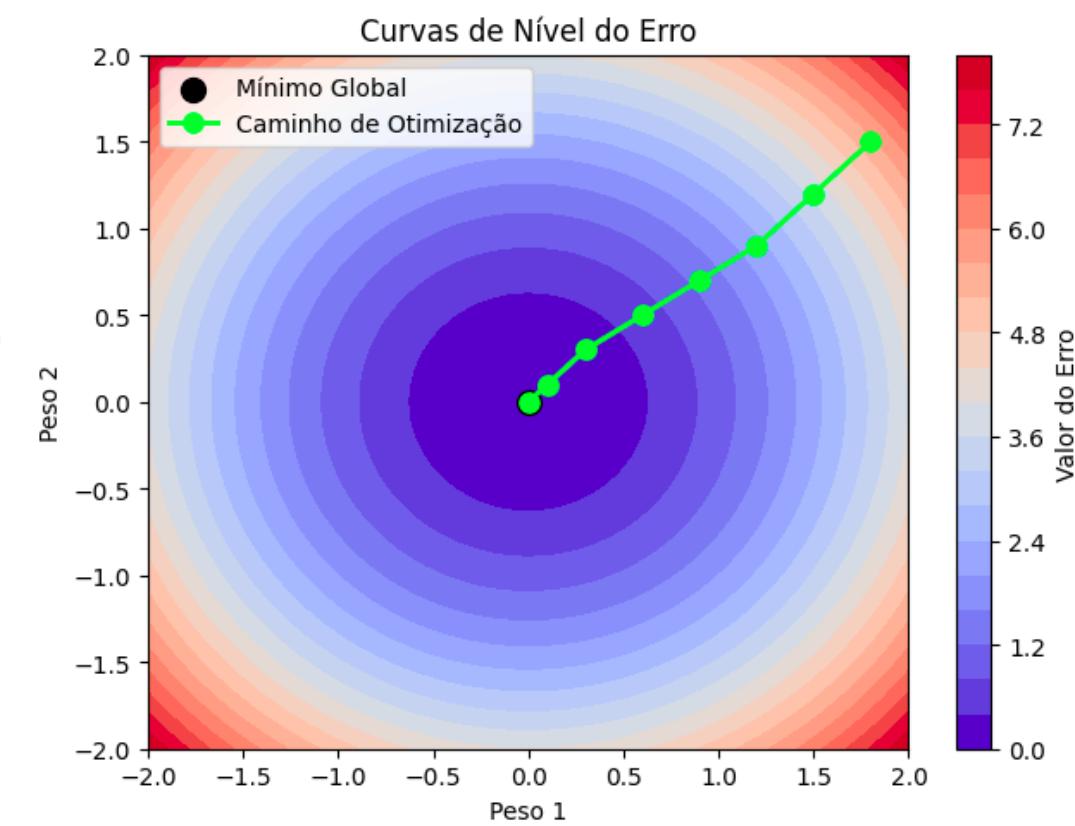
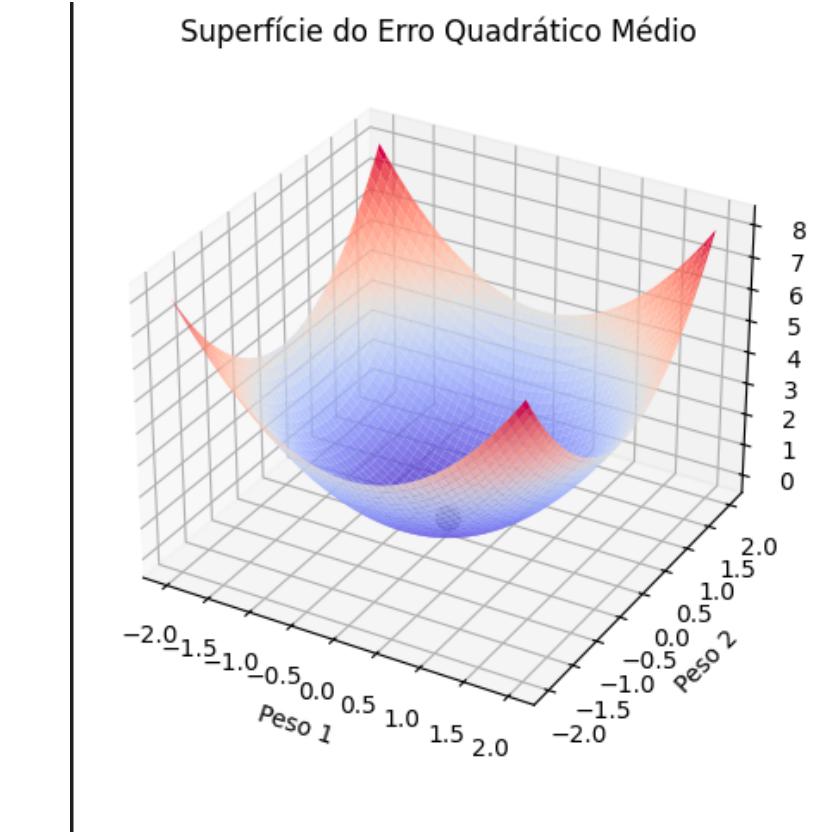
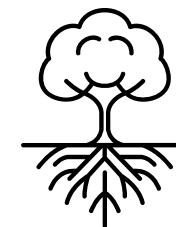
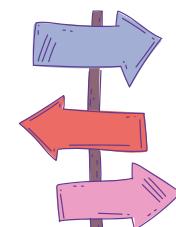


- Critério de parada



Importância do Cálculo do Erro

- Medida de desempenho
- Direção do aprendizado
- Base para o Backpropagation
- Critério de parada



Backpropagation

Princípio da Regra da Cadeia

Se temos uma função composta $E = f(g(h(w)))$, onde w é um peso da rede, a derivada de E em relação a w é dada por:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial f} \cdot \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial w}$$

No contexto de uma rede neural, esta fórmula nos permite calcular como cada peso contribui para o erro total

Backpropagation

Cálculo do Gradiente na Camada de Saída

Para os neurônios na camada de saída, o erro é calculado diretamente comparando a saída prevista com o valor desejado. O gradiente do erro em relação à saída do neurônio é:

$$\delta_j^L = \frac{\partial E}{\partial a_j^L} = \frac{\partial E}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial a_j^L}$$

Onde:

- δ_j^L é o gradiente do erro para o neurônio j na camada de saída L
- a_j^L é a ativação (saída) do neurônio j na camada L
- z_j^L é a soma ponderada das entradas para o neurônio j na camada L

Backpropagation

Cálculo do Gradiente nas Camadas Ocultas

O gradiente para um neurônio em uma camada oculta é:

$$\delta_j^l = \left(\sum_{k=1}^{n_{l+1}} w_{kj}^{l+1} \cdot \delta_k^{l+1} \right) \cdot f'(z_j^l)$$

Onde:

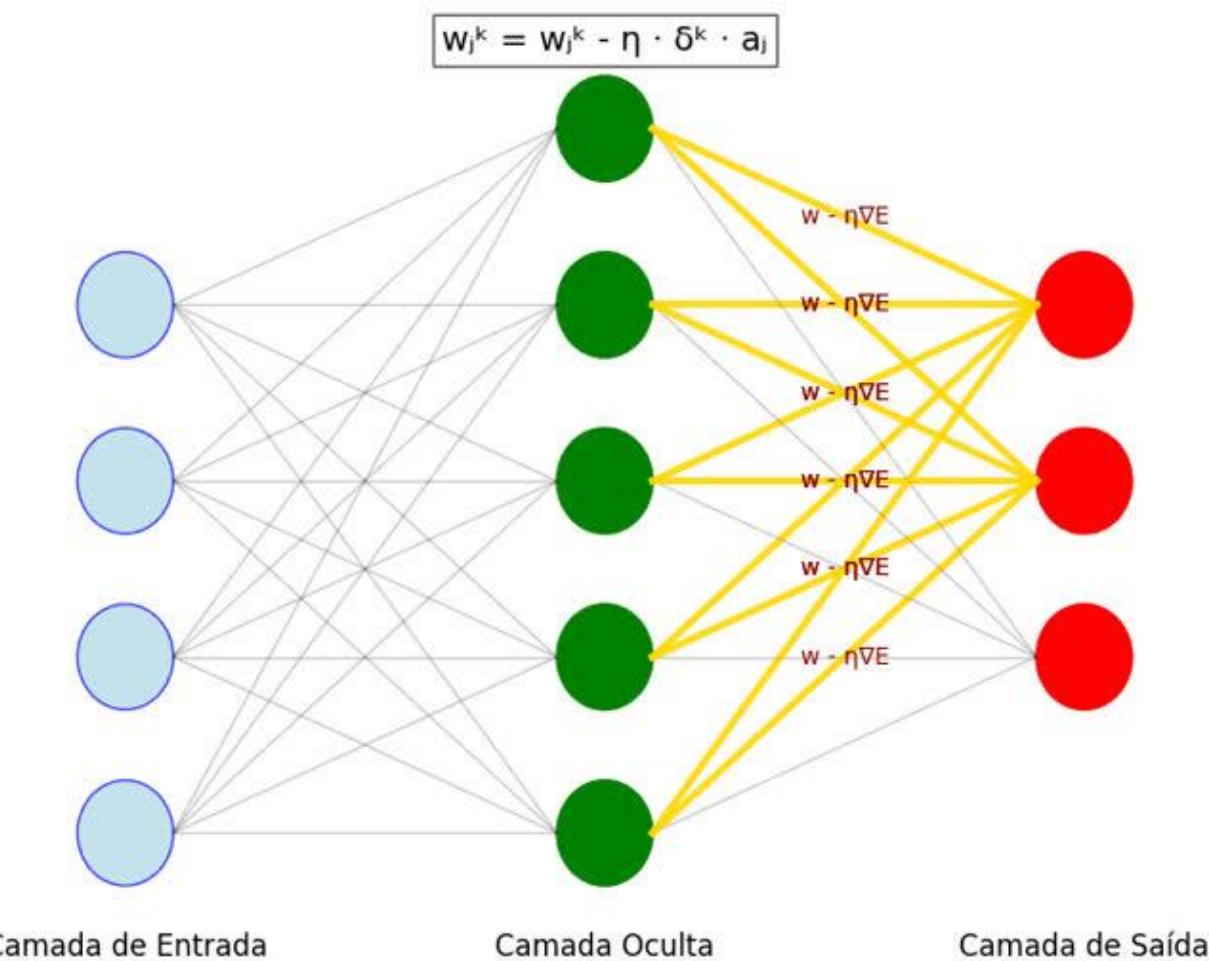
- δ_j^l é o gradiente do erro para o neurônio j na camada l
- w_{kj}^{l+1} é o peso da conexão entre o neurônio j na camada l e o neurônio k na camada $l + 1$
- δ_k^{l+1} é o gradiente do erro para o neurônio k na camada $l + 1$
- $f'(z_j^l)$ é a derivada da função de ativação avaliada em z_j^l
- n_{l+1} é o número de neurônios na camada $l + 1$

Esta fórmula mostra como o erro é propagado de volta através da rede, camada por camada

Backpropagation

Exemplo do Gradiente nas Camadas Ocultas

Atualização dos Pesos: Camada Oculta → Camada de Saída

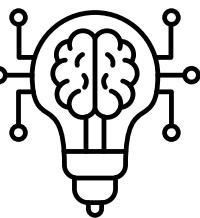


Importância do Backpropagation

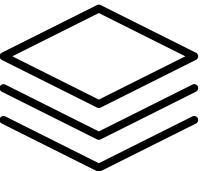
- Atribuição de responsabilidade



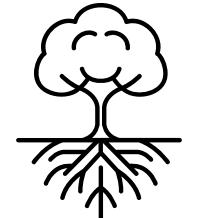
- Aprendizados eficientes



- Aprendizados em camadas profundas



- Base para algoritmos avançados



Atualização dos Pesos

Algoritmo do Gradiente Descendente

No contexto do treinamento de redes neurais, a função a ser minimizada é a função de erro, e os parâmetros são os pesos e vieses da rede. A regra de atualização básica do gradiente descendente é:

$$w'_{ji} = w_{ji} - \eta \frac{\partial E}{\partial w_{ji}}$$

Onde:

- w_{ji} é o peso da conexão entre o neurônio i na camada $l-1$ e o neurônio j na camada l
- η (eta) é a taxa de aprendizado, um hiperparâmetro que controla o tamanho do passo na direção do gradiente
- $\frac{\partial w_{ji}}{\partial E}$ é o gradiente do erro em relação ao peso w_{ji}

De forma similar, os vieses são atualizados usando:

$$b'_j = b_j - \eta \frac{\partial E}{\partial b_j}$$

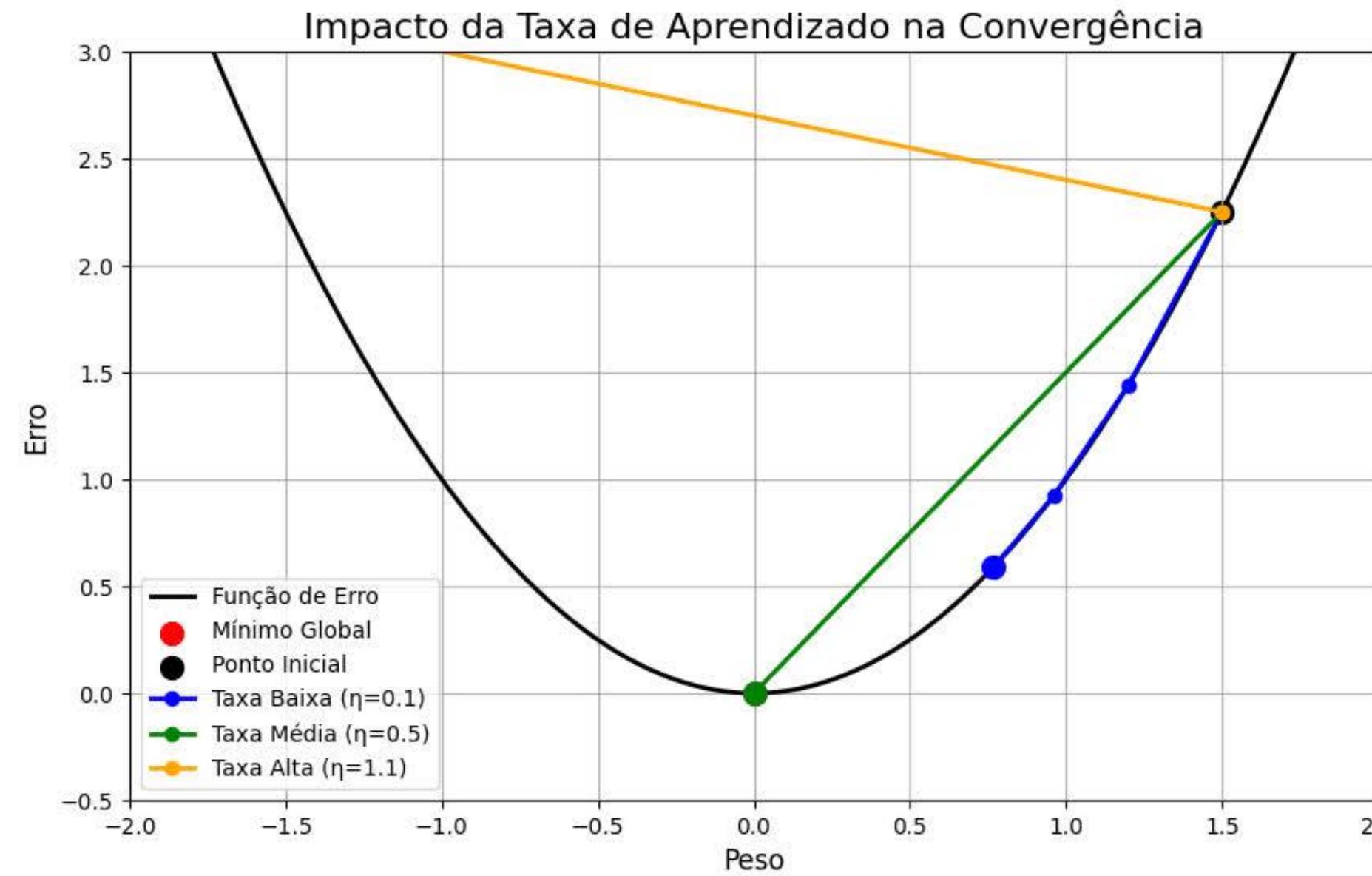
Atualização dos Pesos

Taxa de Aprendizado

- Taxa de aprendizado muito alta: Pode causar oscilações ou divergência, impedindo que o algoritmo encontre o mínimo da função de erro
- Taxa de aprendizado muito baixa: Pode resultar em um treinamento muito lento, exigindo muitas épocas para convergir

Atualização dos Pesos

Exemplo de taxa de Aprendizado



Atualização dos Pesos

Variantes do Gradiente Descendente

1. Gradiente Descendente Estocástico (SGD)

A Ideia: Em vez de calcular o gradiente usando todos os dados (o que é computacionalmente caro), o SGD atualiza os pesos usando apenas um exemplo (ou um pequeno lote) por vez.

$$w = w - \eta \cdot \nabla E(w; x_i, y_i)$$

Onde o gradiente ∇E é calculado para um único exemplo (x_i, y_i).

Prós:

- Muito mais rápido por atualização.
- Exige menos memória.
- O ruído pode ajudar a escapar de mínimos locais.

Contras:

- Atualizações com alta variância (caminho ruidoso).
- Pode "passar reto" do ponto mínimo várias vezes.
- Nunca converge totalmente, fica oscilando ao redor do mínimo.

Atualização dos Pesos

Variantes do Gradiente Descendente

2. Gradiente Descendente com Momentum

A Ideia: Acelerar a descida acumulando uma "velocidade" das atualizações passadas. Ajuda a suavizar o caminho do SGD e a acelerar em direções consistentes.

Fórmulas de Atualização:

- Calcula a nova velocidade (v):

$$v = \gamma v - \eta \cdot \nabla E$$

- Atualiza os pesos com a velocidade:

$$w = w + v$$

Onde γ (gama) é o coeficiente de momentum (geralmente ~ 0.9).

Prós:

- Acelera a convergência.
- Amortece oscilações, resultando em um caminho mais direto.

Contras:

- Adiciona um novo hiperparâmetro (γ) para ser ajustado.

Atualização dos Pesos

Variantes do Gradiente Descendente

3. RMSProp (Root Mean Square Propagation)

A Ideia: Adaptar a taxa de aprendizado (η) individualmente para cada peso. A taxa diminui para pesos que recebem gradientes grandes e aumenta para os que recebem gradientes pequenos.

Fórmulas de Atualização:

- Acumula o gradiente ao quadrado (S):

$$S_{dw} = \beta S_{dw} + (1 - \beta)(\nabla E)^2$$

- Atualiza o peso, dividindo pela raiz do acumulado:

$$w = w - \eta S_{dw} + \epsilon \nabla E$$

Onde β é um fator de decaimento e ϵ (épsilon) é um número pequeno para evitar divisão por zero.

Prós:

- A taxa de aprendizado se ajusta automaticamente.
- Excelente para paisagens de erro complexas (ex: vales longos e estreitos).

Contras:

- Ainda depende de uma taxa de aprendizado global (η).

Atualização dos Pesos

Variantes do Gradiente Descendente

4. Adam (Adaptive Moment Estimation)

A Ideia: Combina o melhor dos dois mundos. Usa o momentum para acumular velocidade (1º momento do gradiente) e o RMSProp para adaptar a taxa de aprendizado por parâmetro (2º momento do gradiente).

Fórmulas de Atualização:

- Calcula o momentum (m), como no **Momentum**.

$$m = \beta_1 m + (1 - \beta_1) \nabla E$$

- Calcula o acumulador de gradiente ao quadrado (v), como no **RMSProp**.

$$v = \beta_2 v + (1 - \beta_2) (\nabla E)^2$$

- Atualiza os pesos usando m e v corrigidos.

$$w = w - \eta v + \epsilon m$$

Prós:

- Combina as vantagens do Momentum e do RMSProp.
- Geralmente converge mais rápido e é mais robusto.
- Considerado o otimizador padrão para muitos problemas.

Contras:

- Mais complexo computacionalmente.

VISUALIZAÇÃO E EXEMPLOS NO IPYNB



EXEMPLO PRÁTICO

**Visualização do Algoritmo de Backpropagation em Redes Neurais
Mostrando todas as etapas do processo de aprendizado**

Backpropagation - Epoca 3

Arquitetura da Rede Neural

Feedforward

Propagação para Frente (Feedforward)

1. Multiplicação de Pesos
2. Adição de Bias

Camada 1
$z^1 = W^1 a^0 + b^1$ $a^1 = \sigma(z^1)$
$z^2 = W^2 a^1 + b^2$ $a^2 = \sigma(z^2)$

Cálculo do Erro (Binary Cross-Entropy)

Binary Cross-Entropy

$$E = - \sum (y \log(p) + (1 - y) \log(1 - p))$$

Backpropagation

Propagação Reversa (Backpropagation)

1. Cálculo dos Deltas
2. Cálculo dos Gradiantes
3. Propagação do Erro

Cálculo dos Deltas

$$\delta^L = (\hat{y} - y) \odot \sigma'(z^L)$$

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

Cálculo dos Gradientes

$$\frac{\partial E}{\partial W^l} = \delta^l (a^{l-1})^T$$

$$\frac{\partial E}{\partial b^l} = \delta^l$$

Valores dos Deltas (Gradiente do Erro)

Camada 1	Camada 2
$[-0.0329, -0.0064, -0.0146, -0.0171]$	$[-0.2168]$

Atualização dos Pesos

Atualização dos Pesos e Biases

$$W^l = W^l - \eta \frac{\partial E}{\partial W^l} \quad b^l = b^l - \eta \frac{\partial E}{\partial b^l}$$

Taxa de aprendizado (η): 0.51

Exemplos de Atualização de Pesos

Camada	Peso Anterior	Gradiente	Peso Atualizado
Camada 1	-0.6080	-0.0261	-0.5947
Camada 2	0.5700	-0.1102	0.6262

$W^1[0,0] = -0.6080 - 0.51 \times -0.0261 = -0.5947$
 $W^2[0,0] = 0.5700 - 0.51 \times -0.1102 = 0.6262$

Evolução da Perda (Loss)

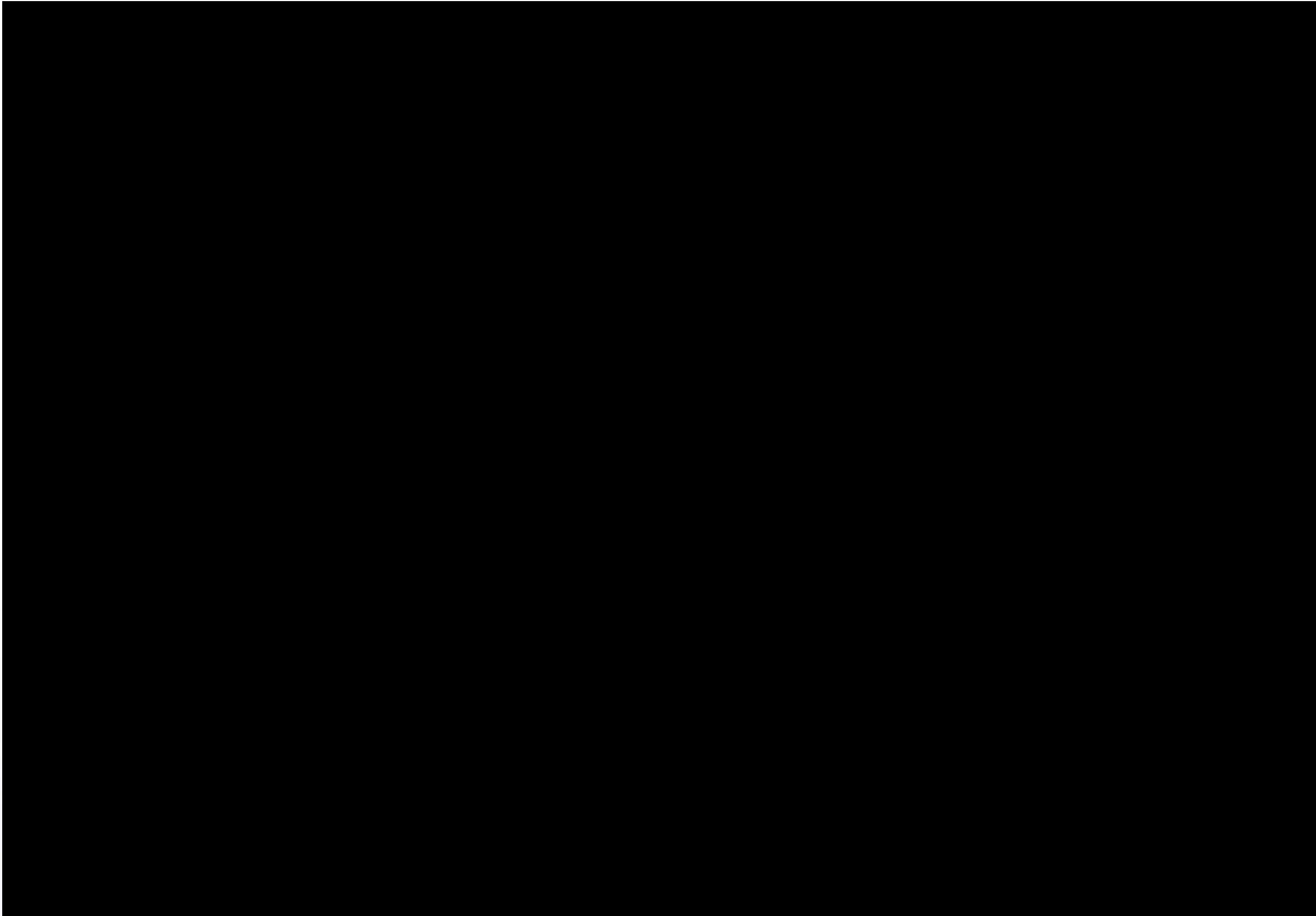
Binary Cross-Entropy

Época

Arquitetura da Rede: [2, 4, 1] | Taxa de Aprendizado: 0.5

Redução do erro: 61.99%

EXEMPLO PRÁTICO



PROBLEMAS NO BACKPROPAGATION E HIPERPARÂMETROS

01 Problemas no Backpropagation

02 Soluções

03 Principais Hiperparâmetros

04 O que são Hiperparâmetros?

PROBLEMAS NO BACKPROPAGATION

- 01 Desvanecimento do Gradiente
- 02 Explosão do Gradiente
- 03 Overfitting (sobreajuste)
- 04 Sensibilidade a Hiperparâmetros



PROBLEMAS NO BACKPROPAGATION

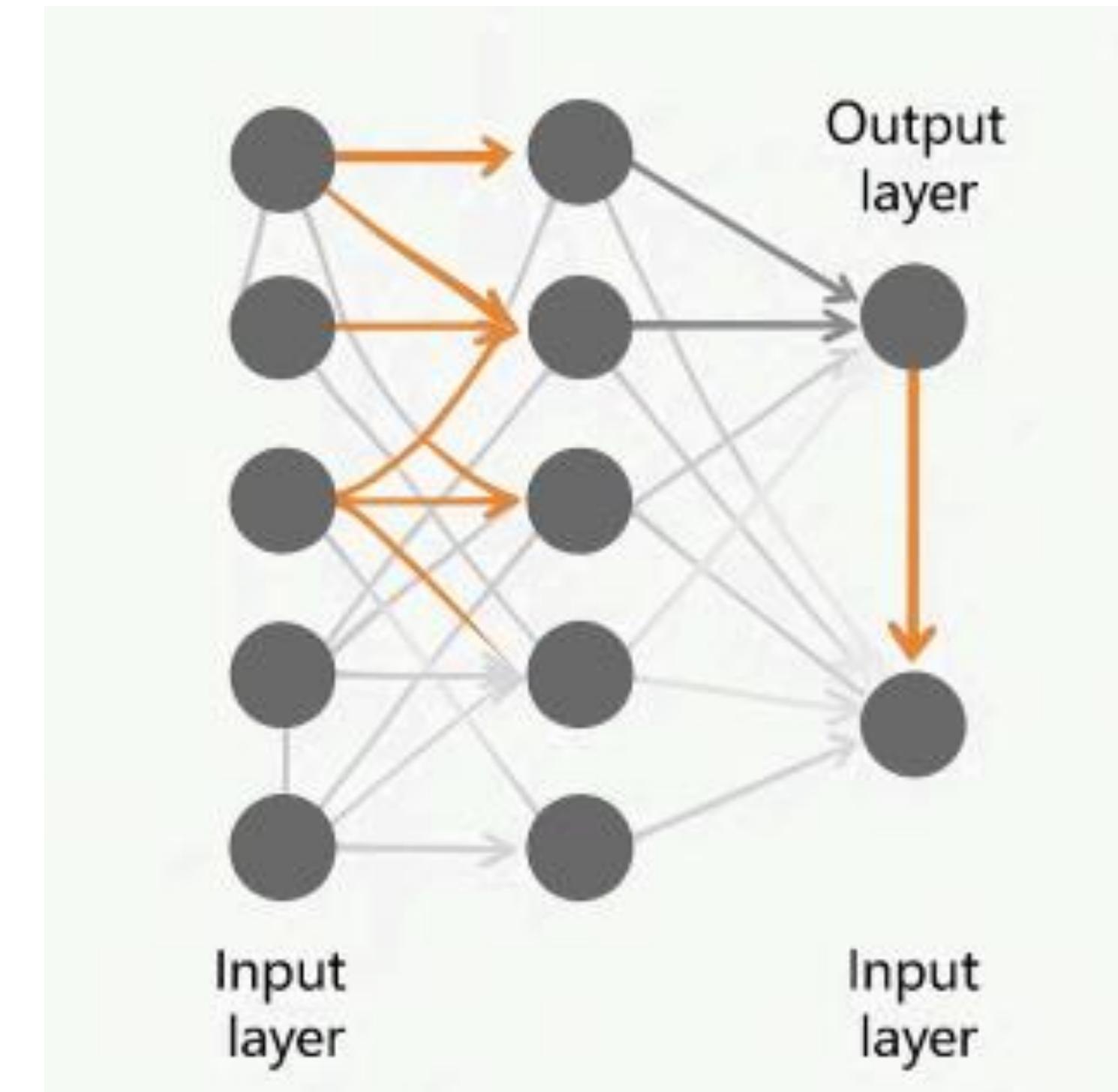
Desvanecimento do Gradiente

- **O que é?**

- Os gradientes se tornam extremamente pequenos à medida que são propagados de volta através das camadas, impedindo o aprendizado efetivo nas camadas iniciais.

- **Por que ocorre?**

- Funções Sigmoide/Hiperbólica: Gradientes máximos de 0.25 e 1.0;
- Redes Profundas: Multiplicação de gradientes pequenos;
- Saturação: Derivadas próximas de zero.



PROBLEMAS NO BACKPROPAGATION

Desvanecimento do Gradiente

- **Consequências:**

- Treinamento extremamente lento;
- Camadas iniciais não aprendem;
- Desempenho subótimo;
- Convergência prejudicada

- **Soluções:**

- ReLU e variantes;
- Batch Normalization;
- Skip Connections (ResNets);
- LSTMs/GRUs

- **Explicação Matemática:**

- Durante o backpropagation, o gradiente em uma camada l é calculado como:
 - $\delta(l) = \delta(l+1) * W(l+1) * f'(z(l))$
 - Se os pesos $W(l+1)$ forem grandes (>1) e/ou a derivada da função de ativação $f'(z(l))$ for grande, os gradientes podem crescer.

PROBLEMAS NO BACKPROPAGATION

Desvanecimento do Gradiente

- **Problema Principal:**

- **Camadas iniciais não conseguem aprender**

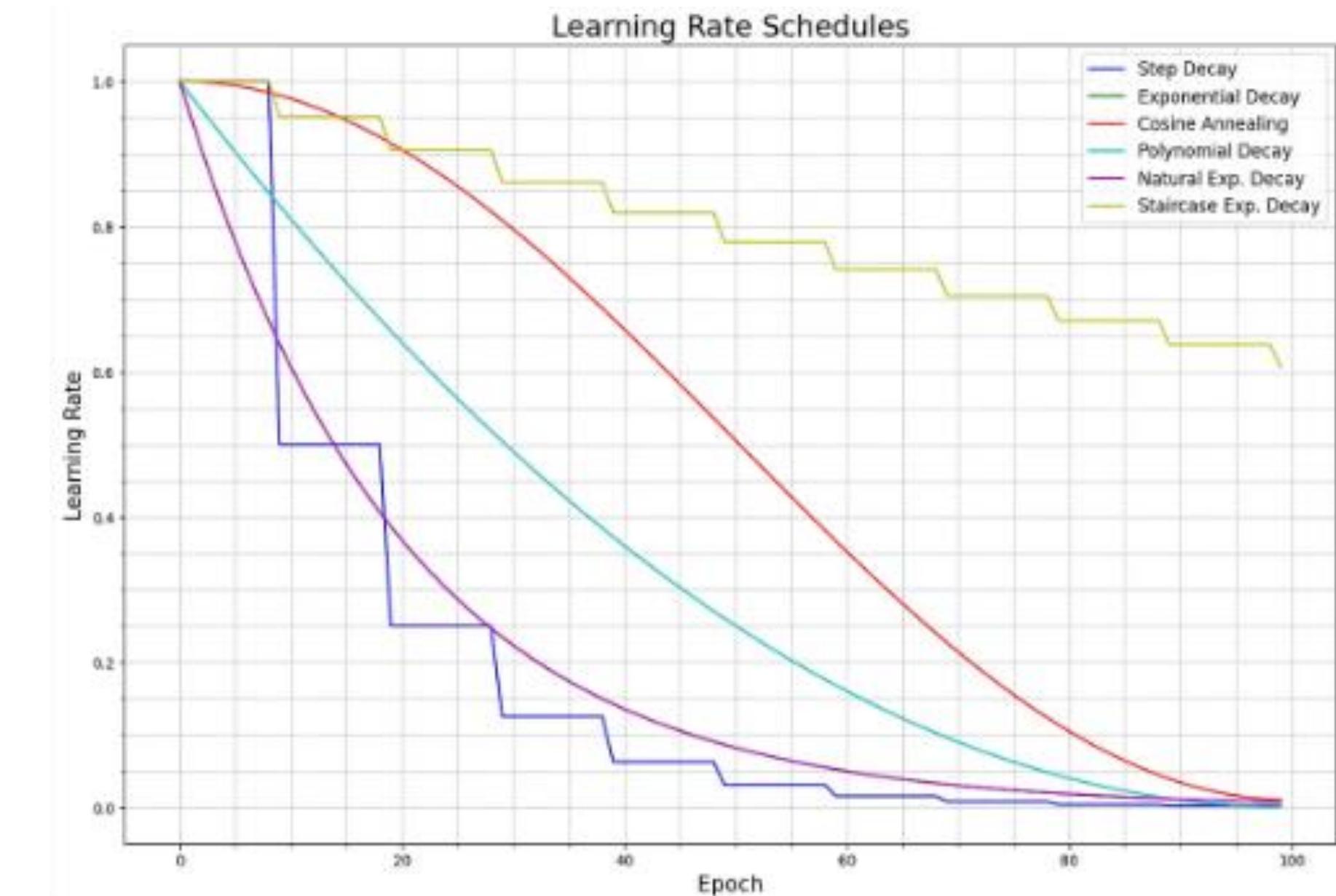
- O desvanecimento do gradiente faz com que os gradientes se tornem exponencialmente menores à medida que se propagam para trás através das camadas da rede neural.

- **Como isso acontece:**

- Os gradientes diminuem drasticamente nas camadas mais próximas à entrada;
 - As atualizações de peso se tornam insignificantes;
 - As camadas iniciais param de aprender efetivamente;
 - A rede não consegue extrair características de baixo nível importantes.

- **Consequência Prática:**

- A rede neural se comporta como se fosse uma rede muito mais rasa, perdendo a capacidade de aprender representações hierárquicas complexas que são a principal vantagem das redes profundas.



PROBLEMAS NO BACKPROPAGATION

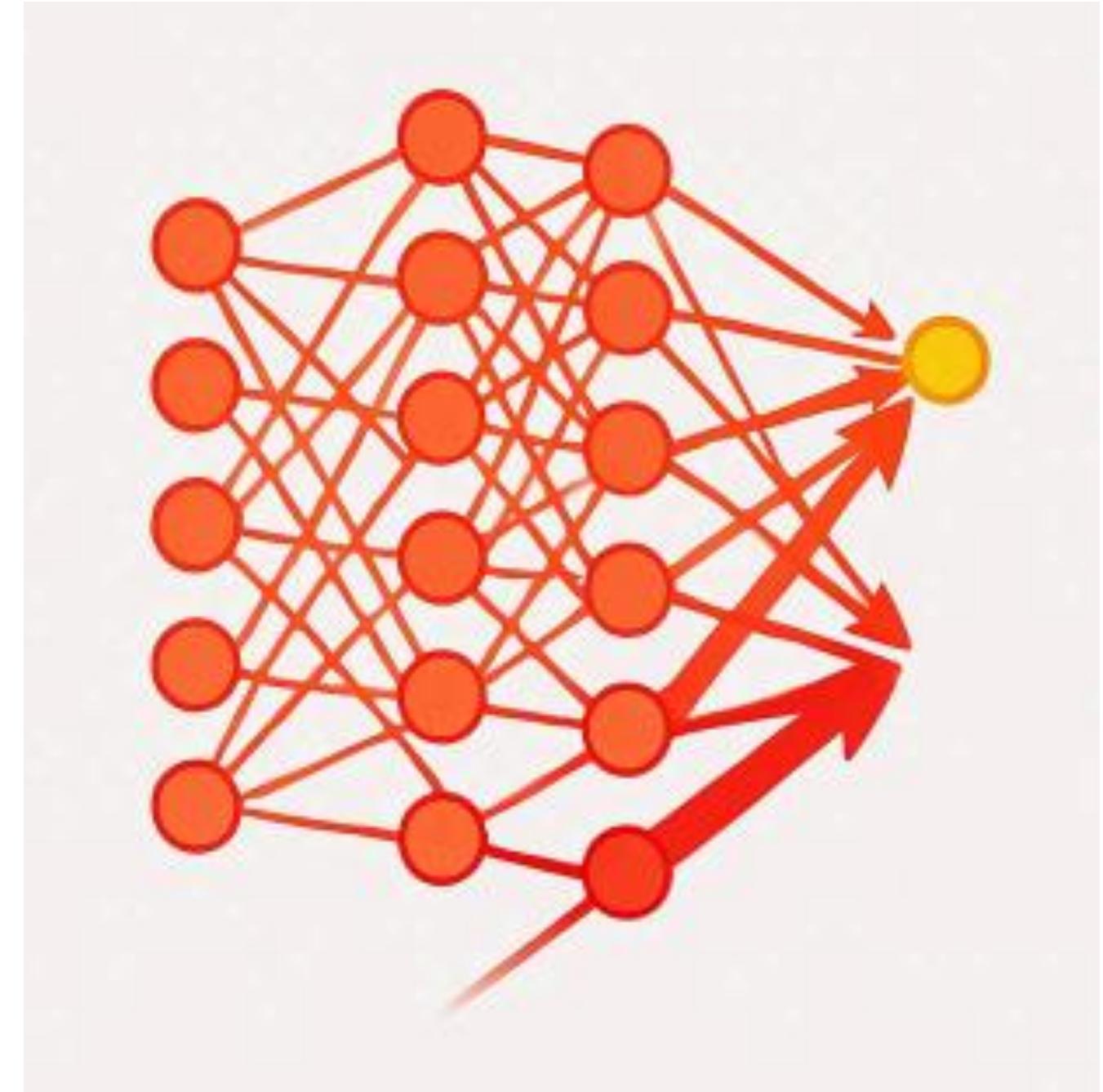
Explosão do Gradiente

- **O que é?**

- Os gradientes se tornam excessivamente grandes durante o Backpropagation, causando atualizações de peso instáveis e potencial divergência do treinamento.

- **Por que ocorre?**

- Pesos Iniciais Grandes: Multiplicação exponencial;
- Taxa de Aprendizado Alta: Amplifica gradientes;
- Redes Profundas: Acumulação de gradientes.



PROBLEMAS NO BACKPROPAGATION

Explosão do Gradiente

- **Consequências:**

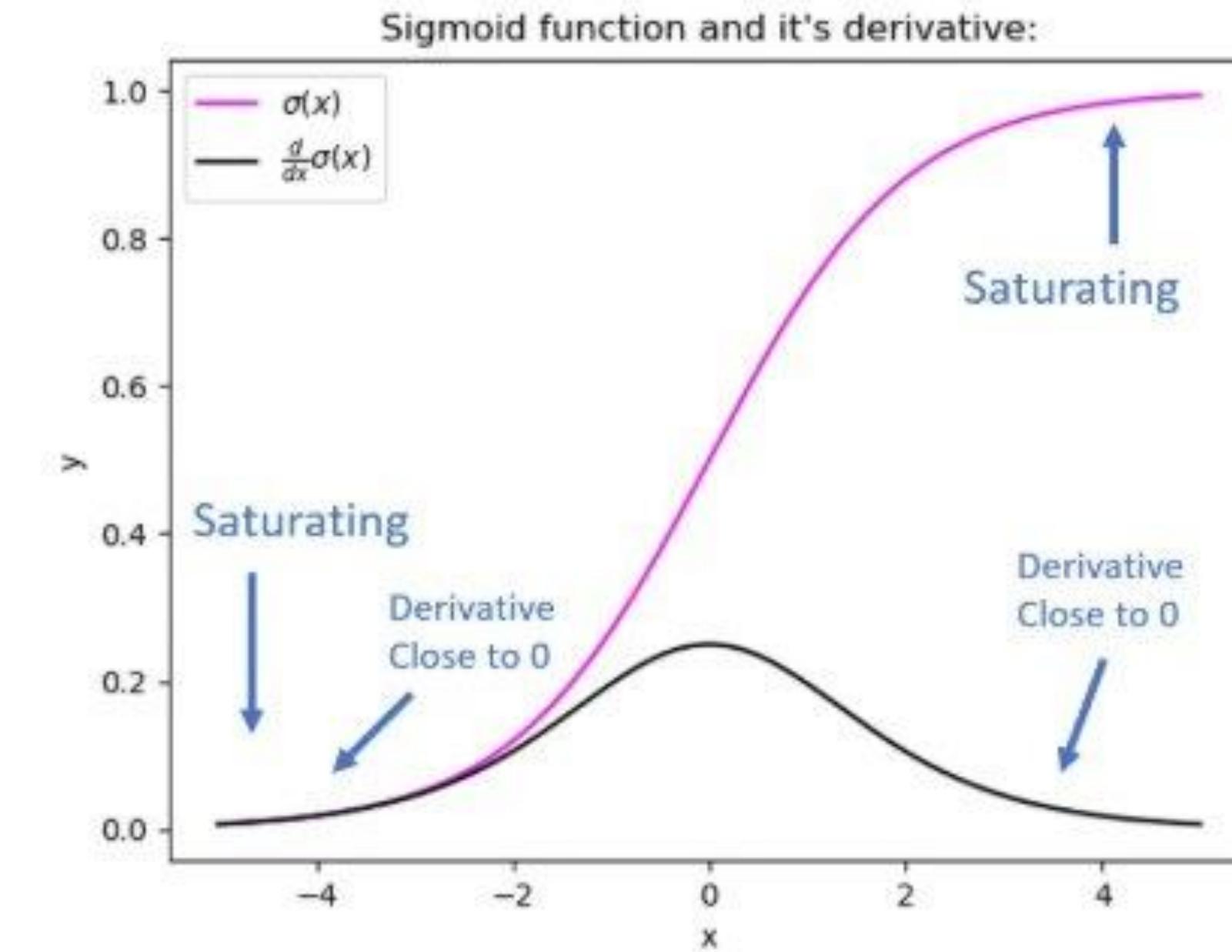
- Flutuações erráticas na perda;
- Valores NaN ou Inf na perda;
- Pesos crescem descontroladamente;
- Instabilidade no treinamento.

- **Soluções:**

- Gradient Clipping: Limita magnitude;
- Regularização L1/L2: Controla pesos;
- Taxa de Aprendizado Menor;
- Batch Normalization.

- **Explicação Matemática:**

- Durante o backpropagation, o gradiente em uma camada l é calculado como:
 - $\delta(l) = \delta(l+1) * W(l+1) * f'(z(l))$
 - Se os pesos $W(l+1)$ forem grandes (>1) e/ou a derivada da função de ativação $f'(z(l))$ for grande, os gradientes podem crescer.



PROBLEMAS NO BACKPROPAGATION

Early Stopping

- **O que é Early Stopping?**

- O Early Stopping é uma técnica de regularização que visa prevenir o overfitting, monitorando o desempenho do modelo em um conjunto de validação e interrompendo o treinamento quando o desempenho começa a piorar.

- **Como Funciona:**

- Monitora uma métrica de desempenho (ex: perda de validação);
 - Se a métrica não melhorar por um número de épocas (paciente), o treinamento é interrompido;
 - O modelo com o melhor desempenho no conjunto de validação é restaurado.

- **Vantagens:**

- Prevenção eficaz de overfitting;
 - Economia de recursos computacionais;
 - Simplicidade de implementação.

PROBLEMAS NO BACKPROPAGATION

Early Stopping

- Por que não é a melhor estratégia para Problemas de Gradiente?

- Embora o Early Stopping seja excelente para combater o overfitting, ele não foi projetado para resolver problemas intrínsecos à dinâmica dos gradientes, como o desvanecimento ou a explosão.

- Limitações:

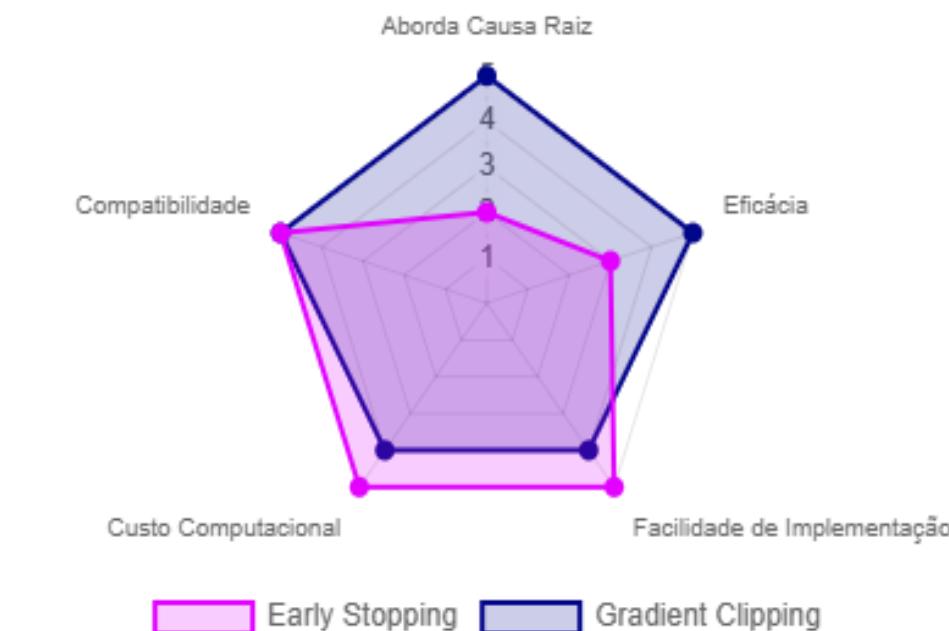
- Não Aborda a Causa Raiz: Problemas de gradiente são inerentes à arquitetura, funções de ativação e inicialização de pesos. Early Stopping não modifica isso;

- Medida de Contenção, não Solução Fundamental: É um "freio de emergência" que impede o modelo de piorar, mas não optimiza o aprendizado para gradientes problemáticos;

- Pode levar a Treinamento Prematuro: Pode interromper o treinamento antes que o modelo atinja seu potencial máximo, especialmente com gradientes desvanecidos.

Técnica	Aborda Causa Raiz	Eficácia	Complexidade
Early Stopping	Não	Baixa	Baixa
Gradient Clipping	Sim	Alta	Baixa
Inicialização de Pesos	Sim	Alta	Baixa
Regularização L1/L2	Parcial	Média	Baixa

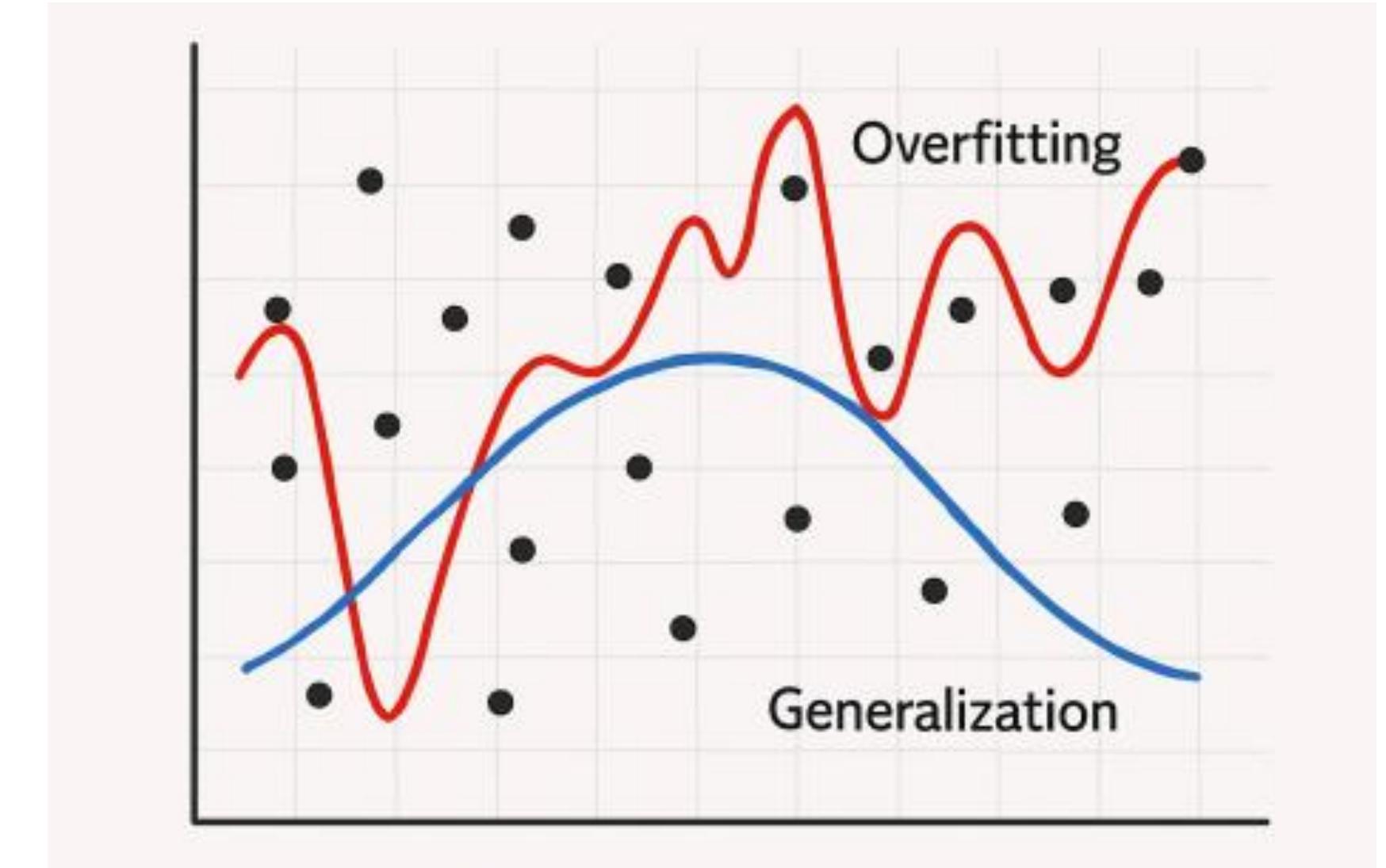
Early Stopping vs. Gradient Clipping



PROBLEMAS NO BACKPROPAGATION

Overfitting (sobreajuste)

- **O que é?**
 - O modelo aprende os dados de treinamento excessivamente, incluindo ruído e particularidades, perdendo a capacidade de generalizar para dados novos.
- **Por que ocorre?**
 - Treinamento Excessivo: Muitas épocas;
 - Modelos Complexos: Alta capacidade;
 - Dados Insuficientes: Memorização.



■ PROBLEMAS NO BACKPROPAGATION

Overfitting (sobreajuste)

- **Como Identificar?**

- Alta precisão no treino, baixa na validação;
- Perda de treino ↓, perda de validação ↑;
- Diferença significativa de performance.

- **Soluções:**

- Regularização L1/L2: Penaliza complexidade;
- Dropout: Desativa neurônios aleatoriamente;
- Early Stopping: Para quando validação piora;
- Data Augmentation: Aumenta diversidade;
- Cross-Validation: Avalia generalização

PROBLEMAS NO BACKPROPAGATION

Sensibilidade a Hiperparâmetros

- **Por que ocorre?**
 - Natureza Iterativa: Backpropagation ajusta pesos em pequenos passos controlados pelos hiperparâmetros;
 - Superfície Complexa: Função de perda não-convexa com múltiplos mínimos locais;
 - Interdependência: Hiperparâmetros interagem entre si de forma complexa.
- **Consequências:**
 - Convergência Lenta/Divergência;
 - Subajuste ou Sobreajuste;
 - Desempenho Subótimo;
 - Instabilidade no Treinamento.
- **Soluções:**
 - Otimização de Hiperparâmetros: Grid Search, Random Search, Bayesian Optimization para encontrar combinações ideais de forma sistemática.

PROBLEMAS NO BACKPROPAGATION

Sensibilidade a Hiperparâmetros

Hiperparâmetros Críticos

- 01** Taxa de Aprendizado
O mais crítico - controla tamanho dos passos
- 02** Função de Ativação Impacta diretamente os gradientes
- 03** Otimizador
Estratégia de atualização dos pesos
- 04** Tamanho do Batch
Afeta estabilidade e convergência

SOLUÇÕES

Agendadores de Taxa de Aprendizado

⌚ O que são?

- Técnicas que ajustam automaticamente a taxa de aprendizado durante o treinamento para melhorar a convergência.

Principais tipos:

- **Step Decay**

- Reduz a taxa por um fator após épocas fixas.

- **Exponential Decay**

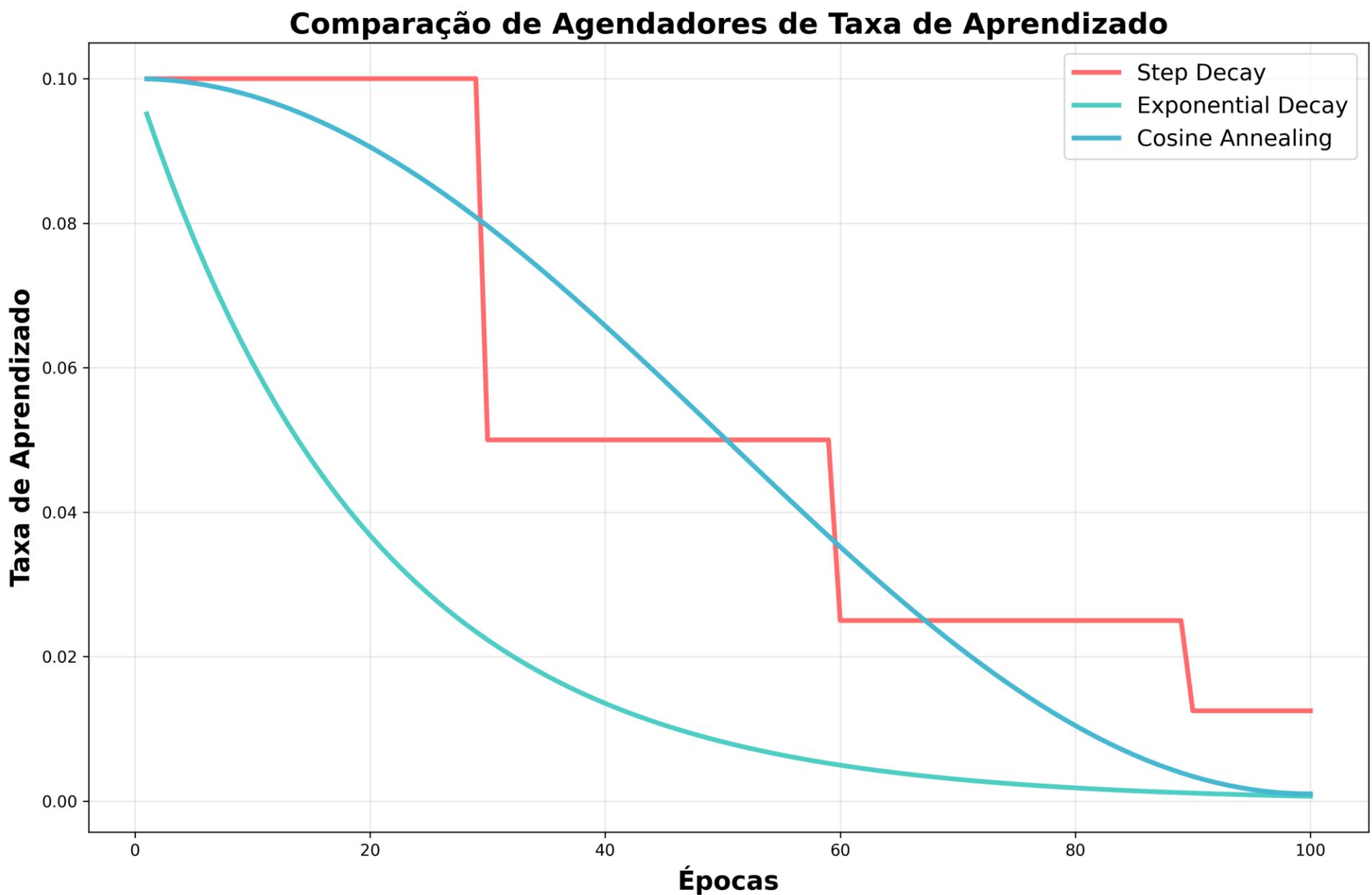
- Diminui exponencialmente a cada época.

- **Cosine Annealing**

- Varia seguindo uma função cosseno.

- **ReduceLROnPlateau**

- Monitora métricas e reduz quando param de melhorar.



SOLUÇÕES

Análise de curva de treino

⌚ O que é?

- Técnica que utiliza a análise visual das curvas de treinamento para identificar problemas e ajustar hiperparâmetros.

• Identificação de Overfitting

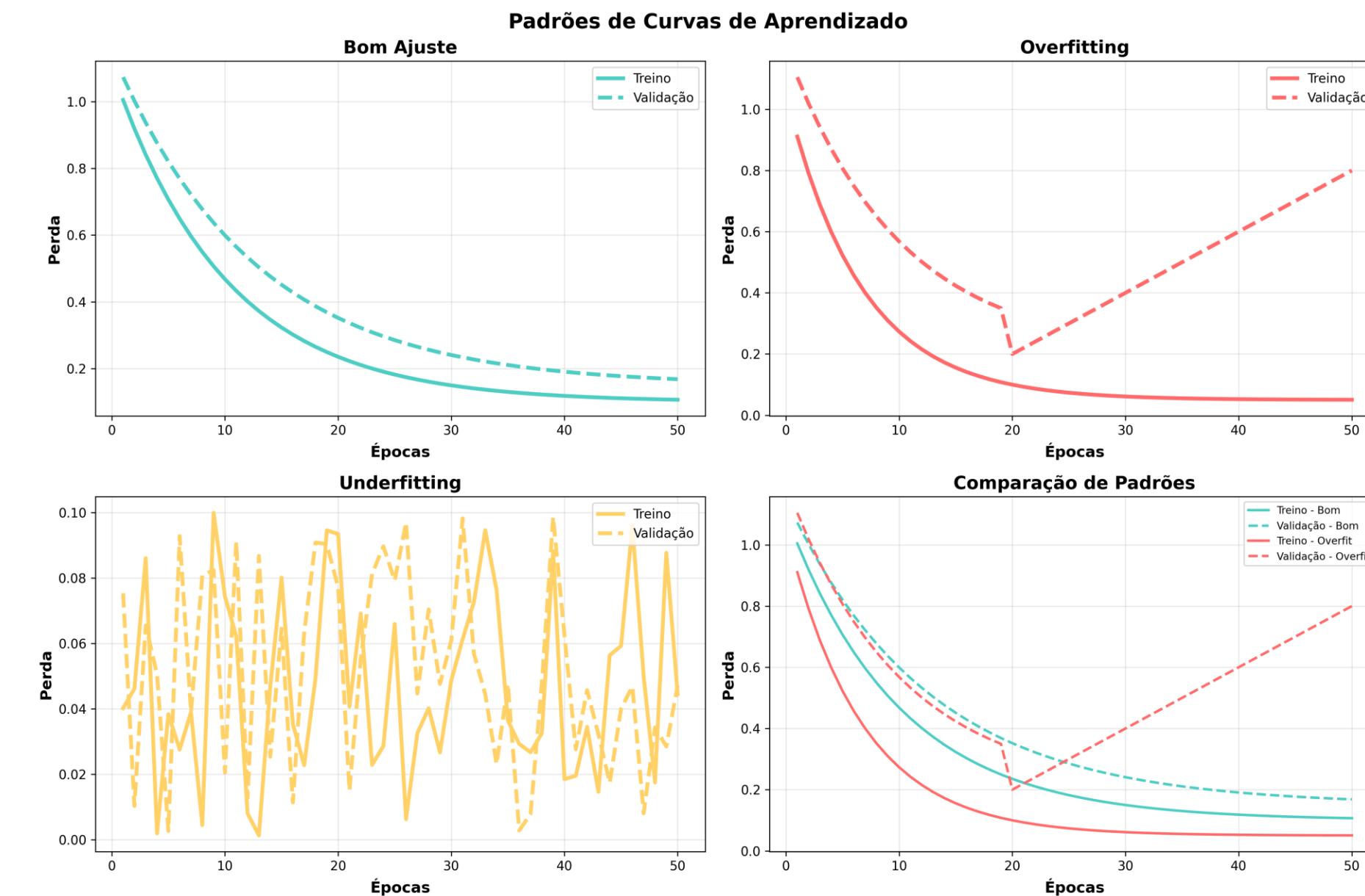
- Perda de treinamento diminui, mas perda de validação aumenta;
- Solução: Regularização, Early Stopping

• Identificação de Underfitting

- Ambas as perdas permanecem altas;
- Solução: Aumentar complexidade do modelo.

• Otimização de Hiperparâmetros

- Validação cruzada;
- Grid Search / Random Search;
- Bayesian Optimization.



SOLUÇÕES

Análise de curva de treino

⌚ O que é?

- Técnica que utiliza a análise visual das curvas de treinamento para identificar problemas e ajustar hiperparâmetros.

• Identificação de Overfitting

- Perda de treinamento diminui, mas perda de validação aumenta;
- Solução: Regularização, Early Stopping

• Identificação de Underfitting

- Ambas as perdas permanecem altas;
- Solução: Aumentar complexidade do modelo.

• Otimização de Hiperparâmetros

- Validação cruzada;
- Grid Search / Random Search;
- Bayesian Optimization.

O QUE SÃO HIPERPARÂMETROS?

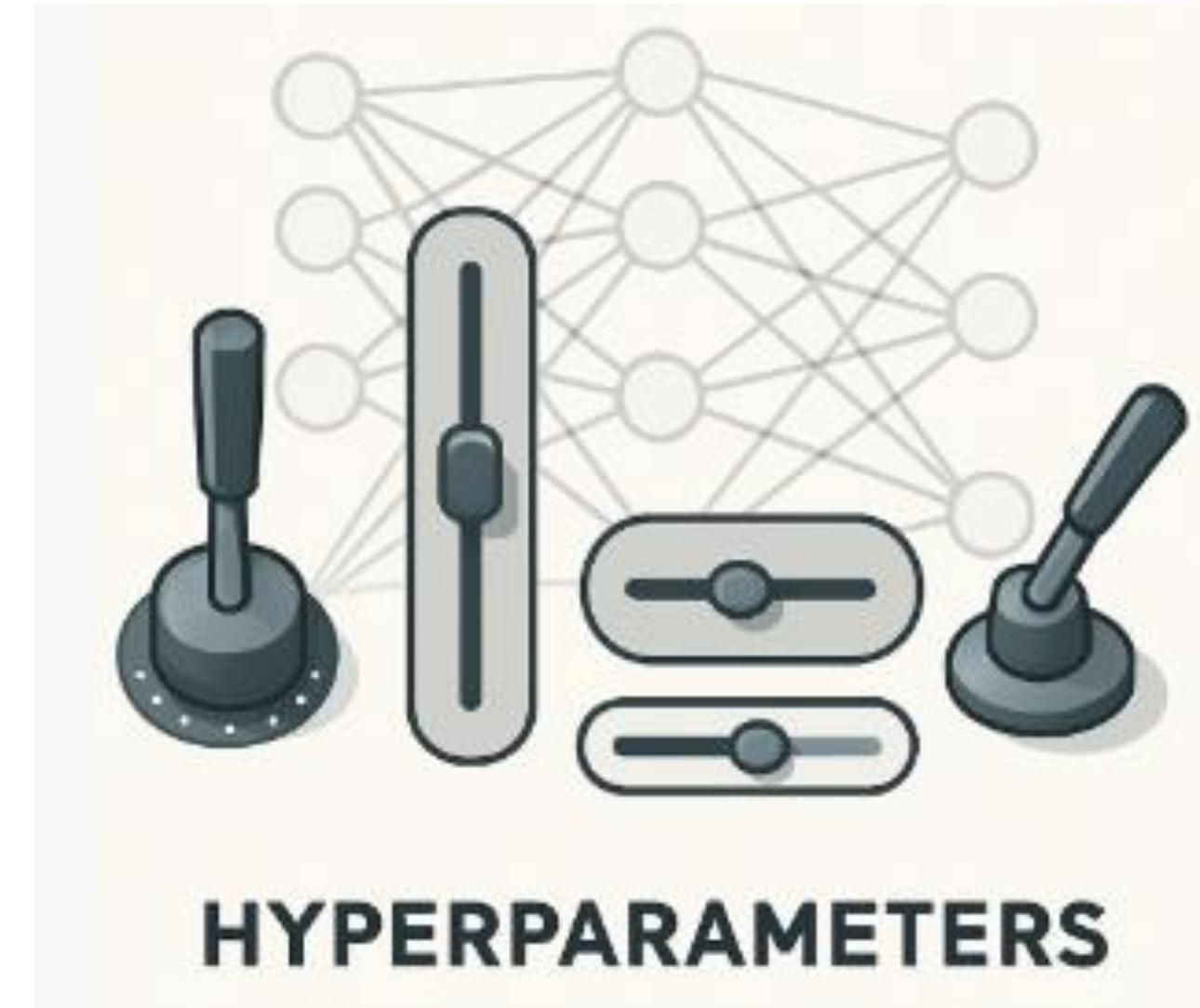
⌚ Definição:

- Variáveis de configuração externas que são definidas antes do treinamento e controlam como o modelo aprende;
- São as "ferramentas" que moldam o processo de aprendizado.

⭐ Por que são importantes?

- Desempenho: Impactam diretamente a precisão do modelo;
- Velocidade: Controlam rapidez do treinamento;
- Complexidade: Definem capacidade do modelo;
- Estabilidade: Influenciam robustez do treinamento.

Enquanto o Backpropagation é o motor que impulsiona o aprendizado, os hiperparâmetros são as alavancas de controle que guiam esse motor.



O QUE SÃO HIPERPARÂMETROS?

Parâmetros vs. Hiperparâmetros

□ PARÂMETROS DO MODELO

- Aprendidos a partir dos dados;
- Pesos e vieses dos neurônios;
- Ajustados automaticamente pelo;
- Backpropagation;
- Resultado do processo de treinamento.

□ HIPERPARÂMETROS

- Definidos pelo usuário antes do treino;
- Taxa de aprendizado, arquitetura • Controlam o processo de treinamento;
- Experimentação e ajuste manual.

O QUE SÃO HIPERPARÂMETROS?

Parâmetros vs. Hiperparâmetros

PROBLEMA

- **PROBLEMA:** Escolha de **Hiperparâmetros Classificação de Imagens**
 - **Cenário:**
 - Você precisa treinar uma CNN para classificar imagens de raios-X em "normal" ou "pneumonia". Como escolher os hiperparâmetros ideais?
 - **Desafio:**
 - Diferentes combinações de hiperparâmetros resultam em:
 - Acurácia variável (70% a 95%);
 - Tempos de treinamento de 1h a 24h;
 - Risco de overfitting ou underfitting.
- **Abordagem:**
 - Grid Search: testar combinações sistemáticas;
 - Random Search: amostragem aleatória do espaço;
 - Bayesian Optimization: aprender com tentativas anteriores.
- **Hiperparâmetros críticos:**
 - Taxa de aprendizado: 0.001 a 0.1;
 - Arquitetura: 3-5 camadas convolucionais;
 - Dropout: 0.2 a 0.5 para evitar overfitting.

PRINCIPAIS HIPERPARÂMETROS

Taxa de Aprendizado (η)

- Controla o tamanho do passo na atualização dos pesos a cada iteração.

↑ ALTA

- ✓ Convergência rápida;
- ✗ Oscilações;
- ✗ Pode divergir;
- ✗ "Salta" sobre mínimo.

↓ BAIXA

- ✓ Convergência estável;
- ✓ Precisão maior;
- ✗ Treinamento lento;
- ✗ Pode travar em mínimos locais.

- Estratégias de Escolha:
 - Experimentação (0.1, 0.01, 0.001);
 - Learning Rate Scheduling;
 - Otimizadores Adaptativos (Adam).

PRINCIPAIS HIPERPARÂMETROS

Número de Épocas

- Uma época = passagem completa de todo o dataset pela rede.

◀ POCAS ÉPOCAS

- ✓ Treinamento rápido;
- ✓ Menor custo computacional;
- ✗ Subajuste (underfitting);
- ✗ Baixa precisão.

▶ MUITAS ÉPOCAS

- ✓ Aprendizado refinado;
- ✓ Alta precisão no treino;
- ✗ Sobreajuste (overfitting);
- ✗ Maior tempo/custo.

- Técnica Principal:
 - Early Stopping:
 - Para o treinamento quando a performance na validação começa a piorar, evitando overfitting..

PRINCIPAIS HIPERPARÂMETROS

Tamanho do Batch

- Número de exemplos processados antes de atualizar os pesos.
 - Batch GD (Batch = Dataset): Gradiente preciso, mas lento para grandes datasets;
 - SGD (Batch = 1) Ruidoso, mas escapa de mínimos locais;
 - Mini-Batch ($1 < \text{Batch} < \text{Dataset}$) Equilíbrio ideal - mais comum na prática

BATCH GRANDE

- ✓ Gradientes estáveis;
- ✓ Melhor paralelização;
- ✗ Mais memória;
- ✗ Mínimos "planos".

BATCH PEQUENO

- Batch Pequeno;
- ✓ Melhor generalização;
- ✓ Menos memória;
- ✗ Gradientes ruidosos ;
- ✗ Convergência instável.

PRINCIPAIS HIPERPARÂMETROS

□ Arquitetura da Rede

- Número de camadas e neurônios por camada definem a capacidade do modelo.

□ REDE SIMPLES

- ✓ Menos overfitting;
- ✓ Treinamento rápido;
- ✗ Pode ser underfitting;
- ✗ Capacidade limitada.

□ REDE COMPLEXA

- ✓ Alta capacidade;
- ✓ Padrões complexos;
- ✗ Risco de overfitting;
- ✗ Mais dados necessários.

- Estratégias de Escolha:
 - Complexidade do Problema: Mais complexo = mais camadas;
 - Volume de Dados: Mais dados = pode usar redes maiores;
 - Experimentação: Começar simples e aumentar gradualmente;
 - Arquiteturas Predefinidas: ResNet, VGG, Transformer.

PRINCIPAIS HIPERPARÂMETROS

Função de Ativação

- Introduz não-linearidade na rede, permitindo aprender relações complexas.

⚠ SIGMOIDE E HIPERBÓLICA

- Historicamente populares;
 - Problema: Desvanecimento do gradiente;
 - Gradientes pequenos em valores extremos;
 - Sigmoid: [0,1], Tanh: [-1,1].
-
- Escolha Prática:
 - Camadas Ocultas: ReLU (padrão);
 - Saída Binária: Sigmoide;
 - Saída Multiclasse: Softmax.

□ RELU E VARIANTES

- ReLU: $f(x) = \max(0, x)$;
- Mitiga desvanecimento do gradiente;
- Problema: Neurônios "mortos";
- Variantes: Leaky ReLU, ELU, PReLU

PRINCIPAIS HIPERPARÂMETROS

Otimizador

- Algoritmo que atualiza os pesos com base nos gradientes calculados.
- SGD: Simples, mas pode ser lento e oscilar;
- Adam: Adaptativo, robusto, escolha padrão;
- RMSprop: Bom para gradientes esparsos;
- Adagrad: Adapta taxa por parâmetro.
- **Características do Adam:**
 - Combina Adagrad + RMSprop;
 - Médias móveis dos gradientes;
 - Menos sensível à taxa de aprendizado;
 - Converge mais rápido.

PRINCIPAIS HIPERPARÂMETROS

Regularização L1/L2

- Adiciona termo de penalidade à função de perda para desencorajar pesos grandes.

$$J_{regularizado}(w) = J_{original}(w) + \lambda \cdot \text{termo de regularização}$$

- **Regularização L1 (Lasso):**

- Penalidade: $\sum |w_i|$;
- Efeito: Alguns pesos $\rightarrow 0$;
- Resultado: Seleção de características;
- Modelo: Esparsos.

- **Regularização L2 (Ridge):**

- Penalidade: $\sum w_i^2$;
- Efeito: "Encolhe" pesos proporcionalmente;
- Resultado: Pesos menores;
- Mais comum em redes neurais.

- **Taxa de Regularização (λ):**

- λ Alto: Forte penalidade, risco de underfitting;
- λ Baixo: Penalidade suave, risco de overfitting.

PRINCIPAIS HIPERPARÂMETROS

Dropout

- "Desliga" aleatoriamente um percentual de neurônios durante o treinamento.

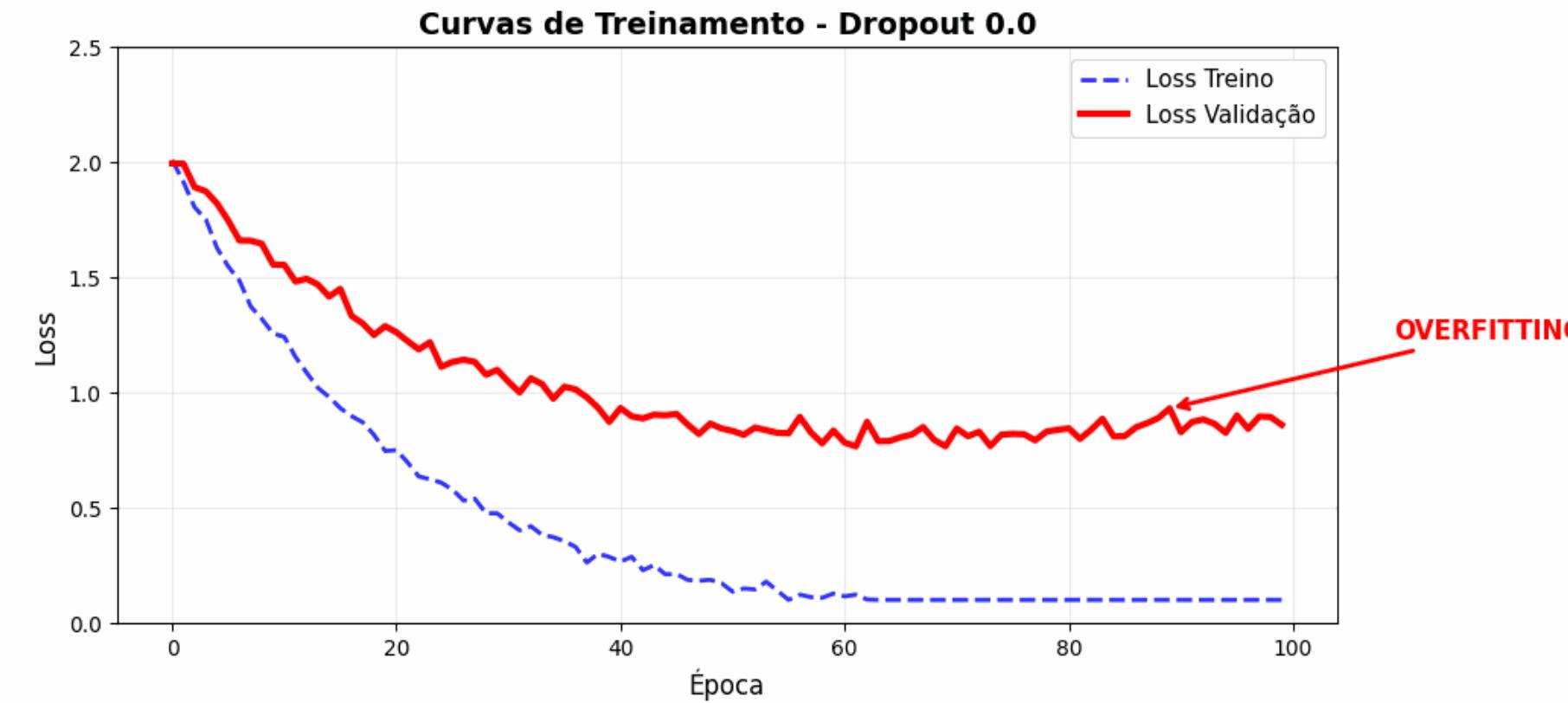
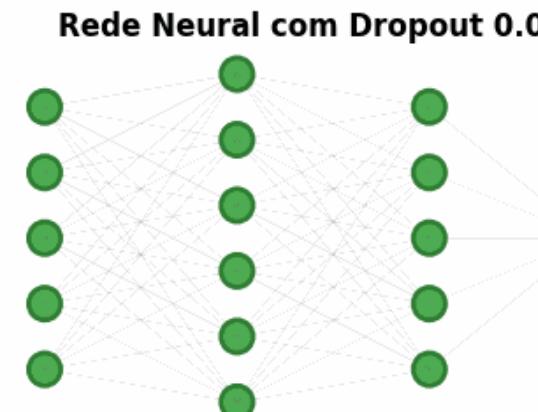
❖ Como Funciona?

- A cada iteração, neurônios diferentes são desativados;
 - Força a rede a não depender de neurônios específicos;
 - Apenas no treinamento - teste usa todos;
 - Pesos são escalados durante inferência.
-
- ↑ **Taxa Alta (ex: 0.7):**
 - ✓ Forte regularização;
 - ✗ Risco de underfitting.
 - ↓ **Taxa Baixa (ex: 0.2):**
 - ✓ Preserva capacidade;
 - ✗ Menos regularização.
 - **Valores Típicos:**
 - Camadas Ocultas: 0.5;
 - Camada de Entrada: 0.2;
 - Experimentar: 0.2 - 0.5.

PRINCIPAIS HIPERPARÂMETROS

Dropout

IMPACTO DO DROPOUT NO OVERFITTING

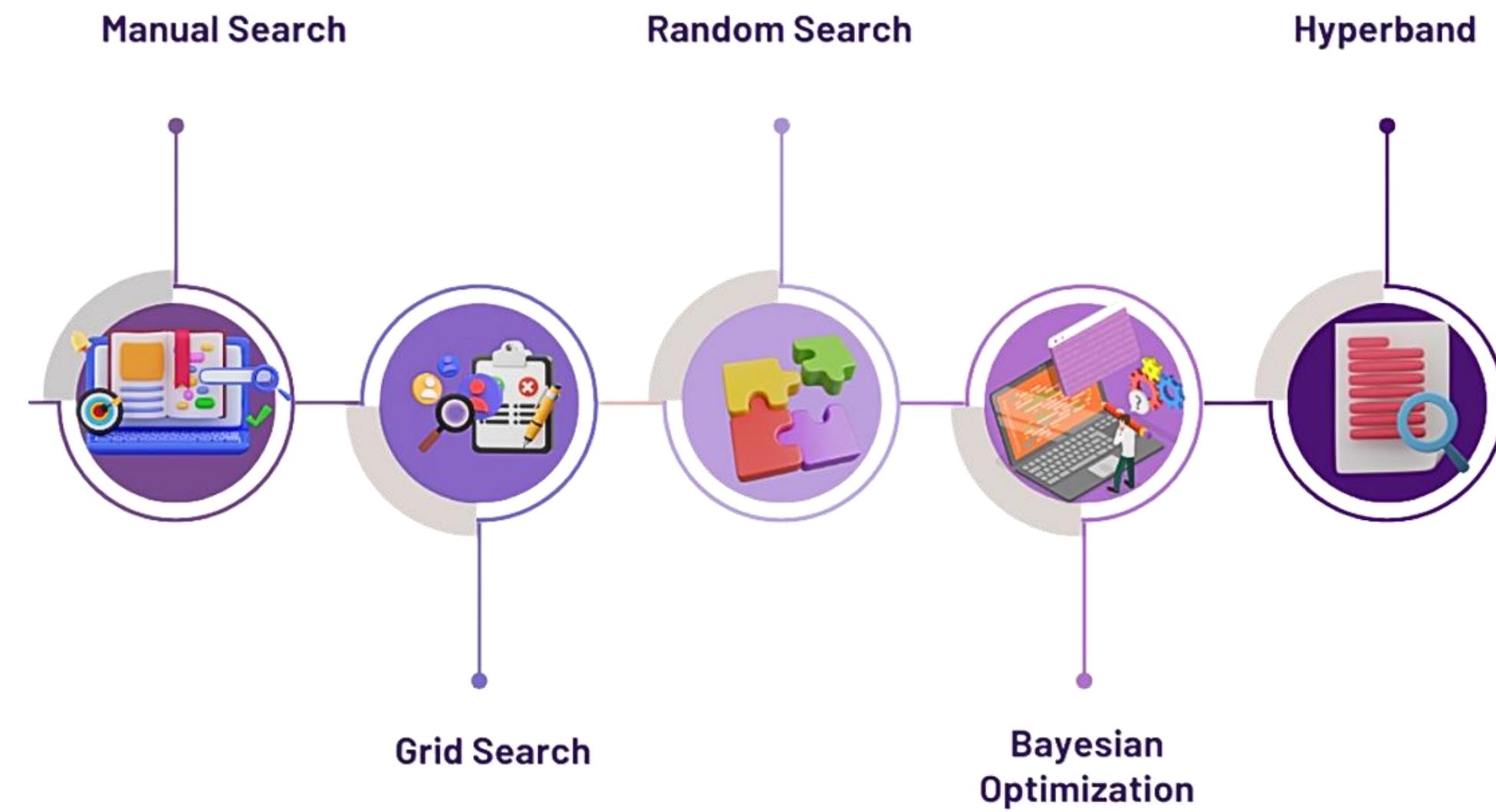


SEM DROPOUT

- Overfitting severo
- Memoriza dados de treino
- Baixa generalização

TAXA BAIXA DEMAIS CAUSA RISCO DE NÃO REDUZIR O OVERFITTING

Técnica para Ajustes Hiperparâmetros



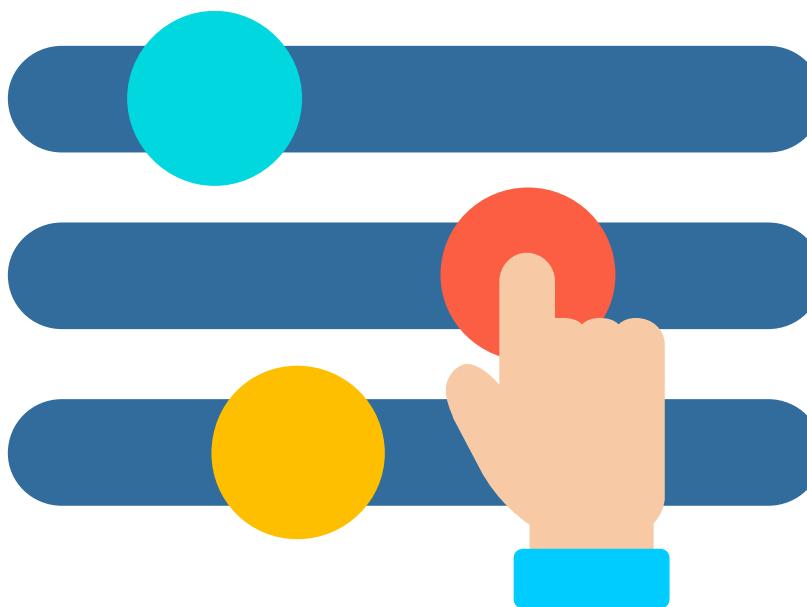
Fonte: AllAboutAi

Técnicas para Ajustes Hiperparâmetros

Manual Search

Definição

- Procedimento a qual utiliza sua experiência, intuição e análises de execuções anteriores para decidir, de **FORMA SUBJETIVA**, quais hiperparâmetros testar em seguida;
- Nesta abordagem, o "motor de otimização" é o próprio ser humano.



Técnicas para Ajustes Hiperparâmetros

Manual Search

8.1.2 Vantagens

- Desenvolvimento baseado na Intuição;
- Flexibilidade;
- Custo de Configuração Zero;
- Potencialmente Rápido (com experiência).

8.1.3 Desvantagens

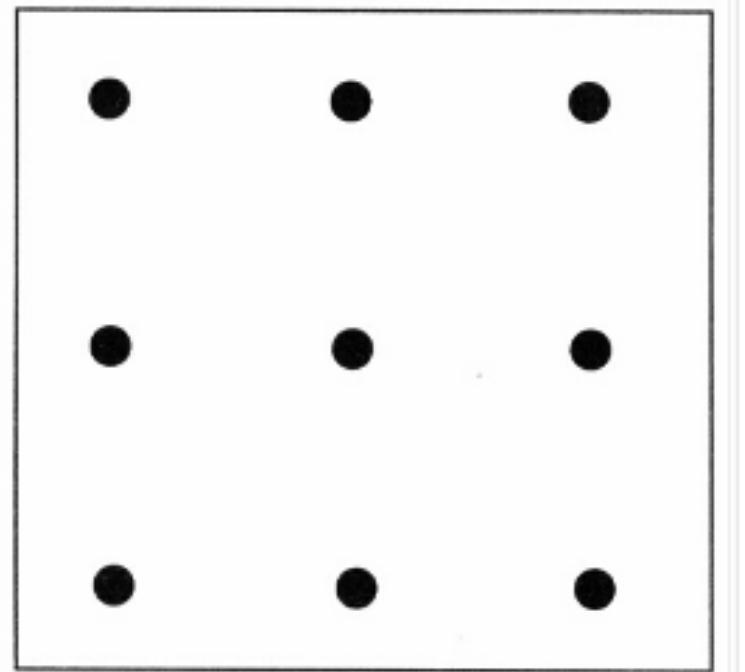
- Não é Cientificamente Reprodutível;
- Propenso a Vieses (Bias);
- Extremamente Lento e Tedioso;

Técnicas para Ajustes Hiperparâmetros

Grid Search

Definição

Avalia **TODAS AS COMBINAÇÕES** de hiperparâmetros a partir de uma grade de valores predefinida manualmente.



Fonte: DataCamp

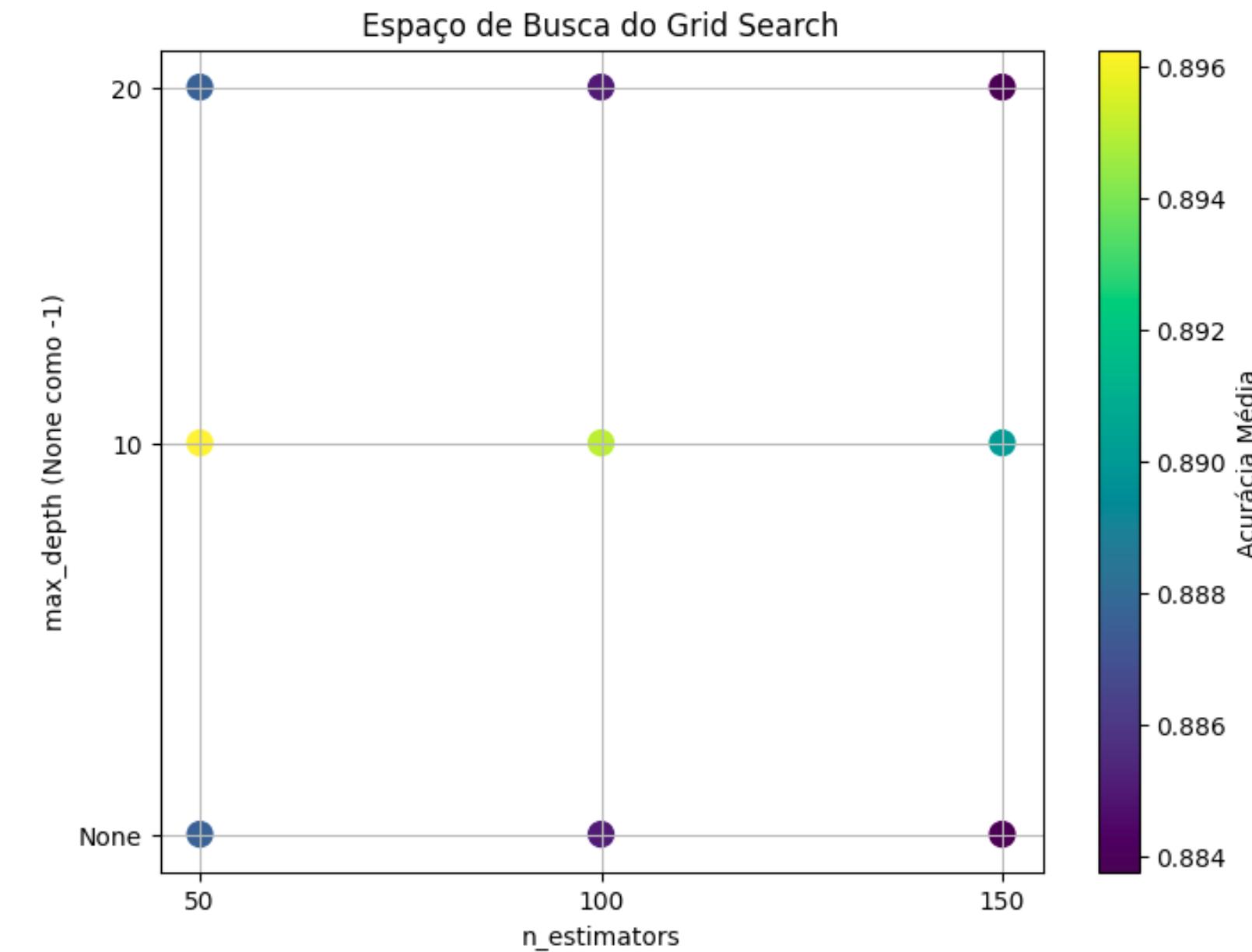
Fundamentação Matemática

- Considere dois hiperparâmetros, λ_1 com os valores $\{a, b\}$ e λ_2 com os valores $\{c, d, e\}$;
- O Grid Search analisa o produto cartesiano dos conjuntos de valores:
- Combinações = $\Lambda_1 \times \Lambda_2 = \{(a, c), (a, d), (a, e), (b, c), (b, d), (b, e)\}$.

Técnicas para Ajustes Hiperparâmetros

Grid Search

Representação Gráfica



Fonte: Autoria Própria

Técnicas para Ajustes Hiperparâmetros

Grid Search

Vantagens

- Simplicidade;
- Paralelização.

Desvantagens

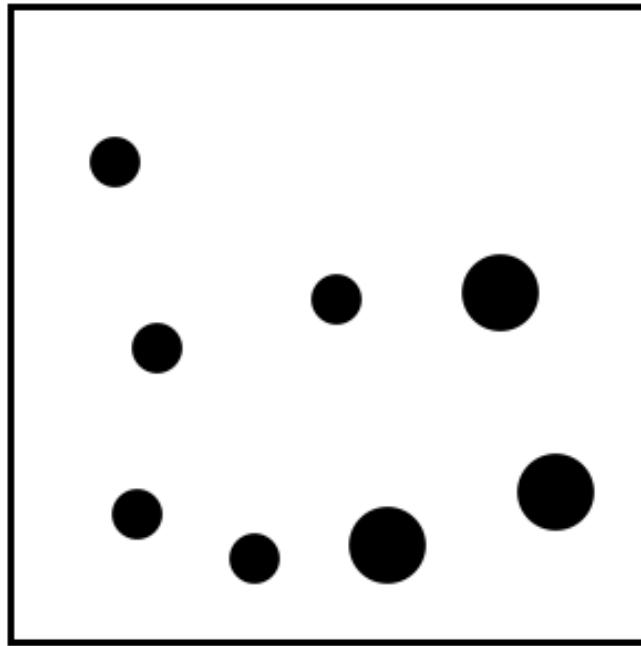
- Custo Computacional;
- Maldição da Dimensionalidade;
- Definição da Grade.

Técnicas para Ajustes Hiperparâmetros

Random Search

Definição

Avalia **UM NÚMERO FIXO** de combinações aleatoriamente a partir de distribuições estatísticas.



Fonte: DataCamp

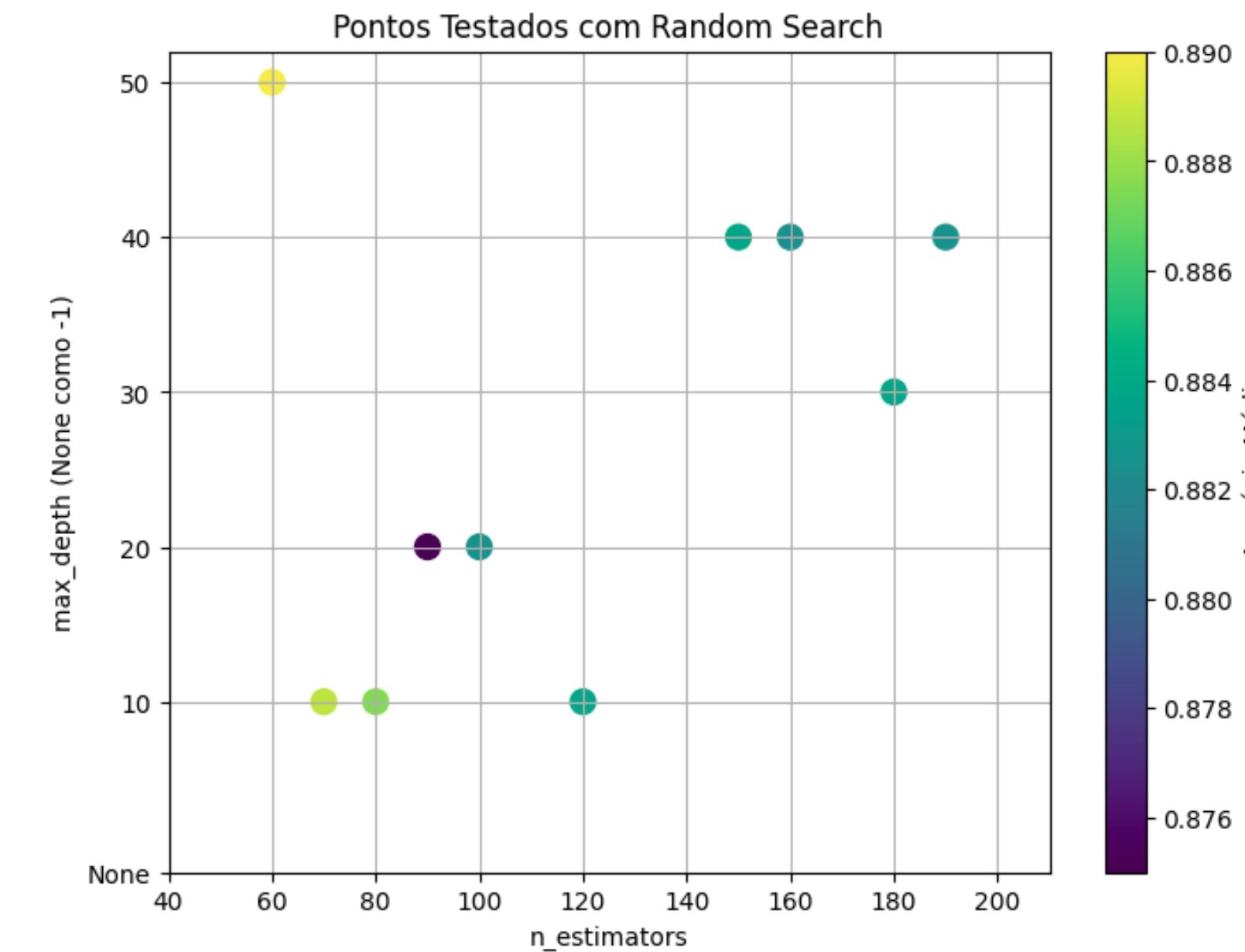
Fundamentação Matemática

- Para cada avaliação, uma configuração λ é extraída de uma distribuição de probabilidade que abrange o espaço de hiperparâmetros Λ ;
- $\lambda(i) \sim p(\lambda | \Lambda)$.

Técnicas para Ajustes Hiperparâmetros

Random Search

Representação Gráfica



Fonte: Autoria Própria

Técnicas para Ajustes Hiperparâmetros

Random Search

Vantagens

- Flexibilidade e Eficiência;
- Controle de Custo;
- Fácil de Paralelizar;
- Garantias Teóricas.

Desvantagens

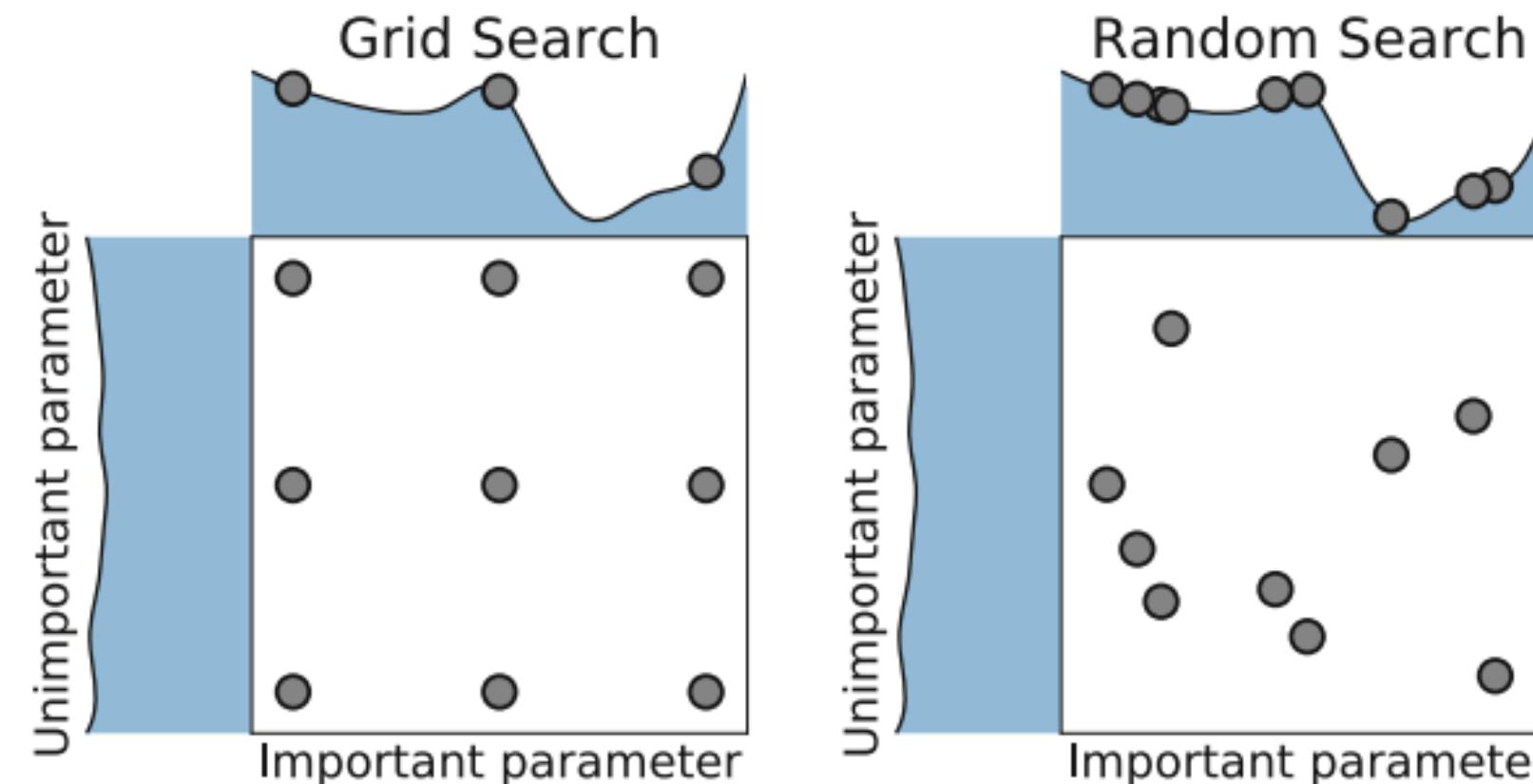
- Não aprende com o processo;
- Ainda pode ser caro.

Técnicas para Ajustes Hiperparâmetros

Random Search

Comparação com o Grid Search

Comparison of grid search and random search for minimizing a function with one important and one unimportant parameter.



Fonte: HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). Automated Machine Learning: Methods, Systems, Challenges. Cham: Springer, 2019.

Técnicas para Ajustes Hiperparâmetros

Bayesian Optimization

Definição

Técnica de otimização sequencial e baseada em modelo. Ela "aprende" sobre a função objetivo (desempenho vs. hiperparâmetros) a cada avaliação e usa esse aprendizado para escolher a próxima combinação a ser testada.

Componentes Chave

- **Modelo Substituto (Surrogate Model):** Uma aproximação probabilística da função objetivo.
Ex.: Processos Gaussianos (GP);
- **Função de Aquisição (Acquisition Function):** Define qual ponto deve ser avaliado em seguida, equilibrando exploração (exploration) e exploração (exploitation).
Ex: Expected Improvement (EI).

Técnicas para Ajustes Hiperparâmetros

Bayesian Optimization

Fundamentação Matemática (Simplificada)

- **Processo Gaussiano (GP):** aproxima a função objetivo (ex: erro do modelo), evitando avaliações lentas e caras.

$$f(\lambda) \sim GP(m(\lambda), k(\lambda, \lambda'))$$

$m(\lambda)$: função média;
 $k(\lambda, \lambda')$: kernel de covariância (semelhança).

Usa um Kernel que assume que configurações similares têm resultados parecidos, permitindo prever o desempenho em áreas ainda não exploradas.

Técnicas para Ajustes Hiperparâmetros

Bayesian Optimization

Fundamentação Matemática (Simplificada)

- **Expected Improvement (EI):** Determina o ganho esperado ao selecionar um novo ponto λ , em comparação com o melhor valor observado f_{\min} .

$$EI(\lambda) = E[\max(f_{\min} - y, 0)]$$

O objetivo é maximizar $EI(\lambda)$ para decidir qual configuração deve ser avaliada a seguir.

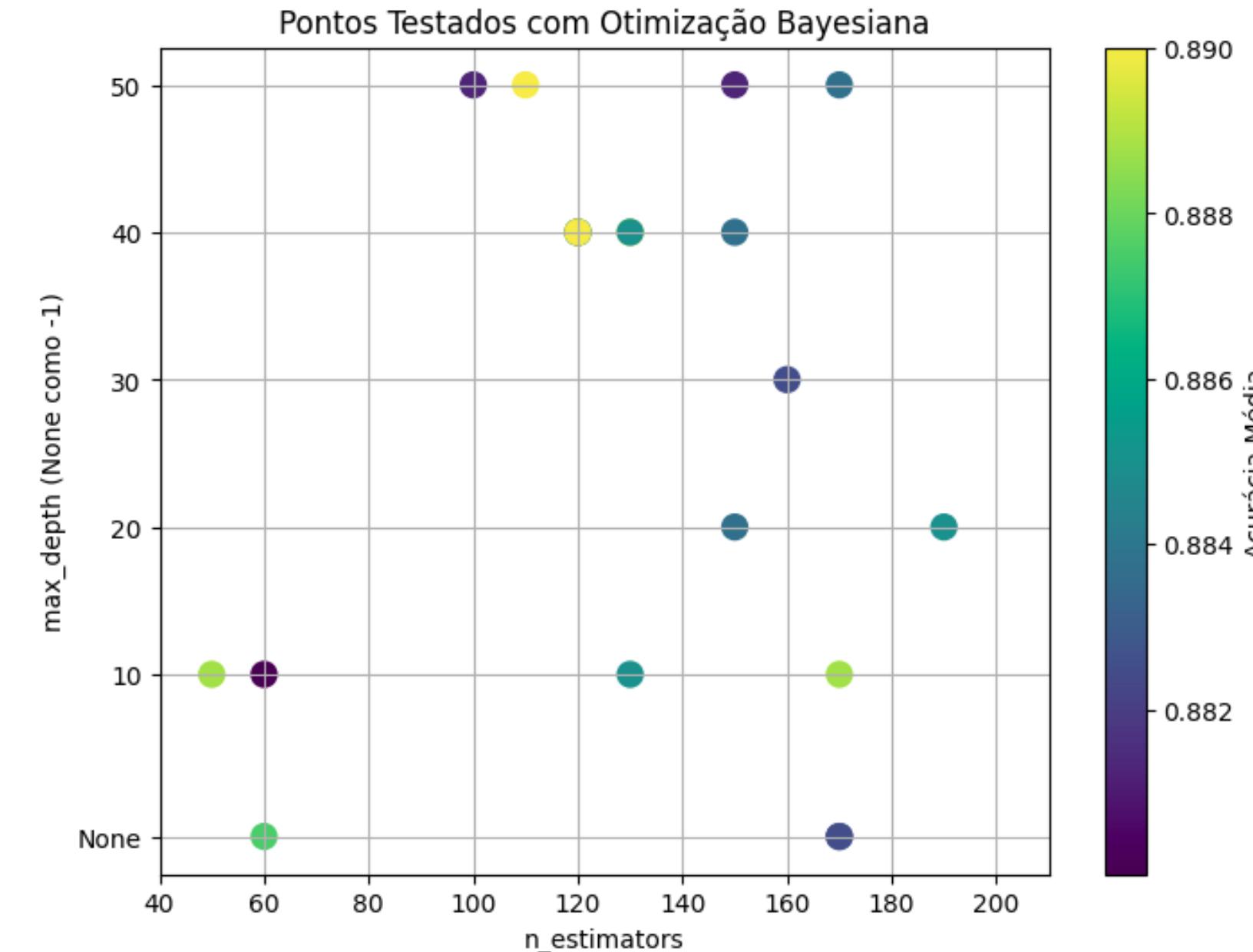
Para cada configuração, calcula o ganho esperado que teríamos em relação ao melhor resultado já encontrado (f_{\min}). Um EI alto significa que o ponto tem a melhor combinação de: Previsão promissora (aproveitamento) e Alta incerteza (exploração)

Técnicas para Ajustes Hiperparâmetros

Bayesian Optimization

Representação

Gráfica



Fonte: Autoria Própria

Técnicas para Ajustes Hiperparâmetros

Bayesian Optimization

Vantagens

- Eficiência de Dados;
- Busca Inteligente;
- Desempenho Superior.

Desvantagens

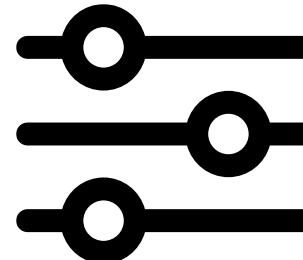
- Complexidade de Implementação;
- Custo de Overhead;
- Dificuldade de Paralelização;
- Risco de Mínimos Locais.

Técnicas para Ajustes Hiperparâmetros

Hyperband

Definição

Uma abordagem de alta velocidade baseada no conceito de Successive Halving (Metade Sucessiva). Ele aloca um orçamento (ex: épocas, subconjunto de dados) a um grande número de configurações e elimina iterativamente as de pior desempenho, focando os recursos nas mais promissoras.



Técnicas para Ajustes Hiperparâmetros

Hyperband

Como Funciona (Algoritmo)

P1 - Começa com n configurações aleatórias;

P2 - Treina todas por um recurso mínimo r (ex: 1 época);

P3 - Elimina a metade com pior desempenho;

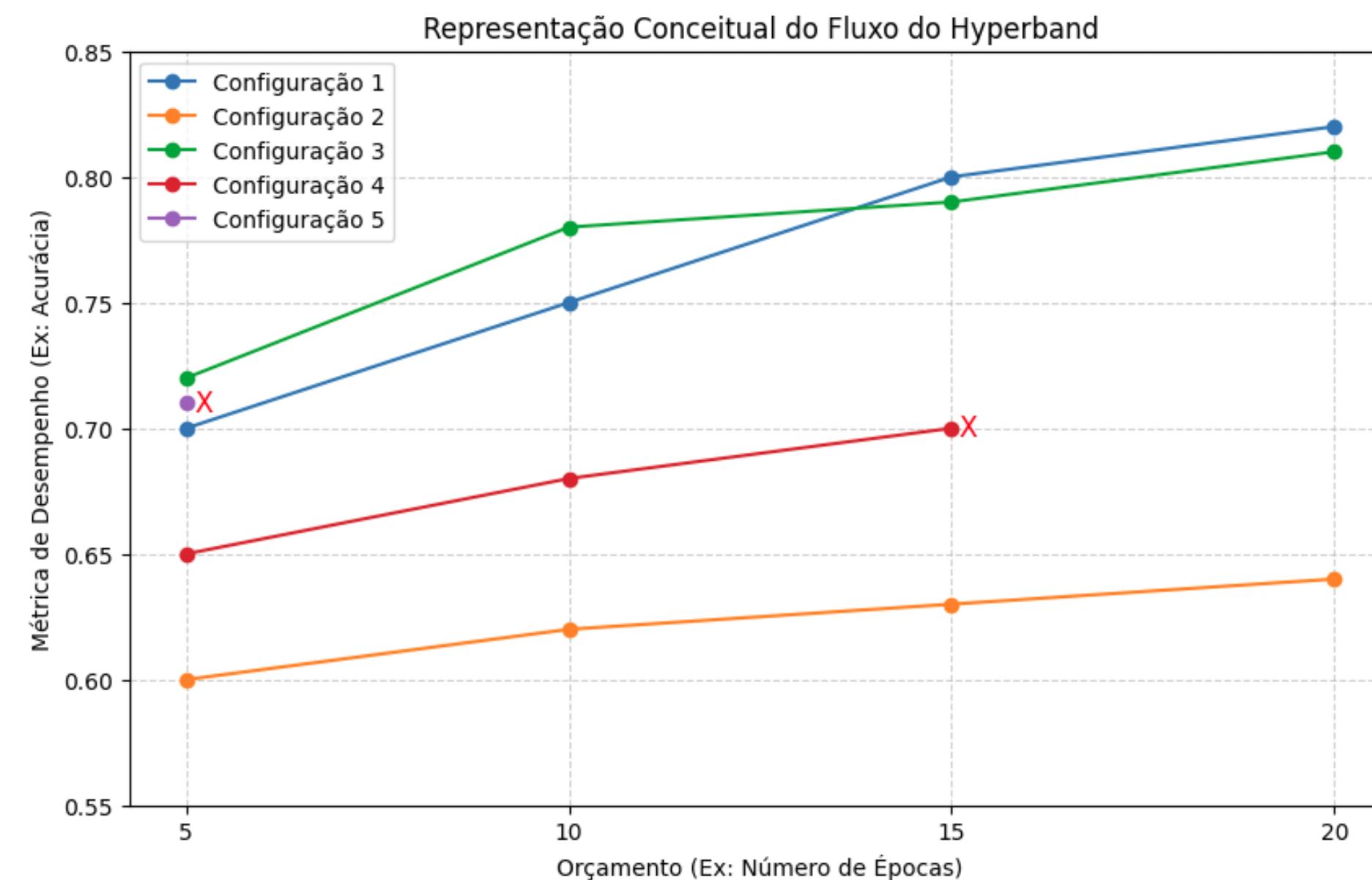
P4 - Dobra o recurso ($2r$) para as sobreviventes e repete até que reste uma única configuração;

P5 O Hyperband repete esse processo com diferentes valores iniciais de n para mitigar o dilema "poucas configurações bem treinadas vs. muitas mal treinadas".

Técnicas para Ajustes Hiperparâmetros

Hyperband

Representação Conceitual



Fonte: Autoria Própria

Técnicas para Ajustes Hiperparâmetros

Hyperband

Vantagens ✓

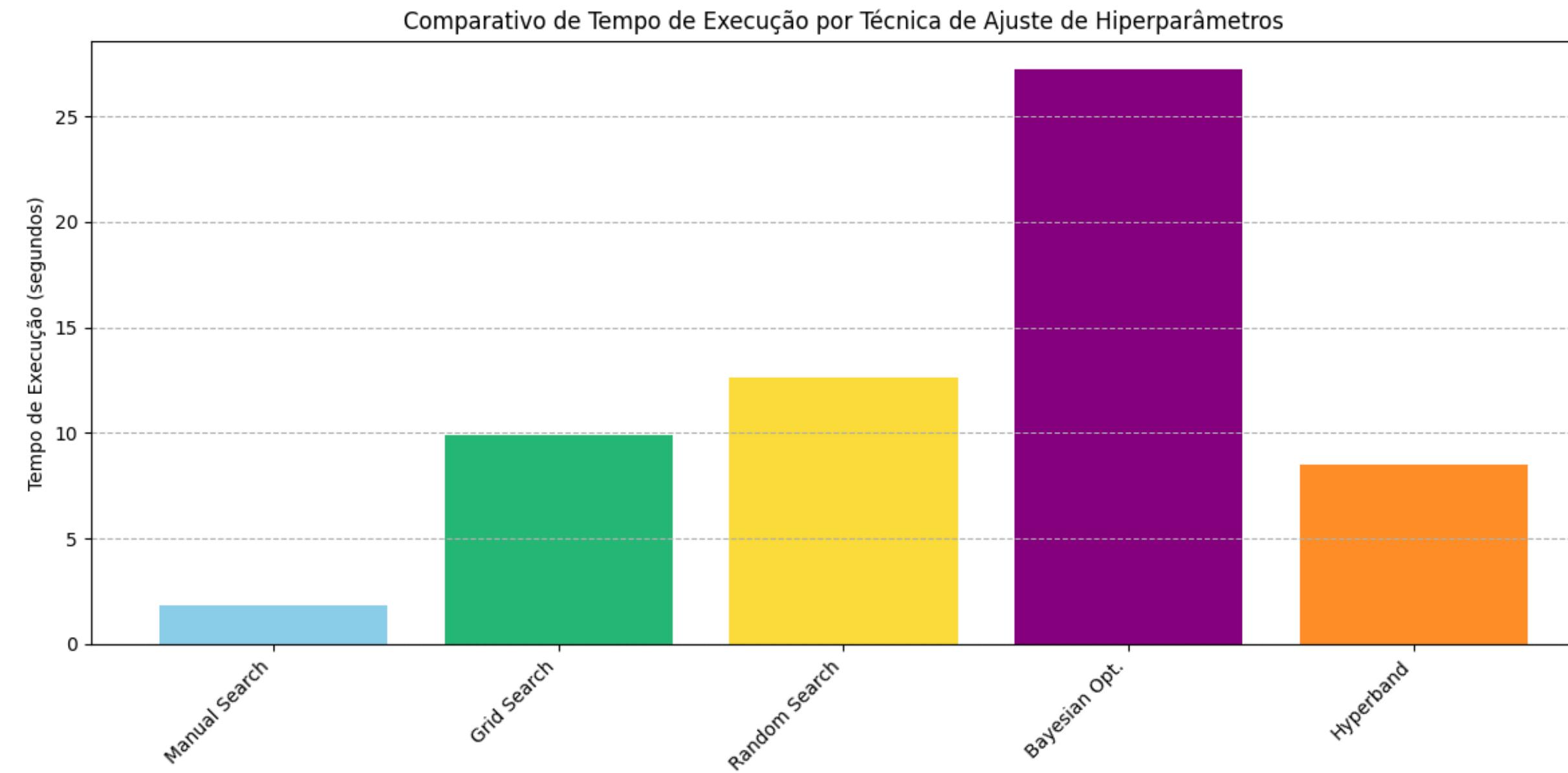
- Velocidade e Eficiência;
- Simplicidade Conceitual;
- Ideal para Deep Learning.

Desvantagens ✗

- Geração por Amostragem Aleatória (Não Adaptativa);
- Risco de Eliminação de modelos com convergência lenta que poderiam se tornar os melhores a longo prazo;
- Requer modelos que permitam avaliações parciais (multi-fidelidade).

Técnicas para Ajustes Hiperparâmetros

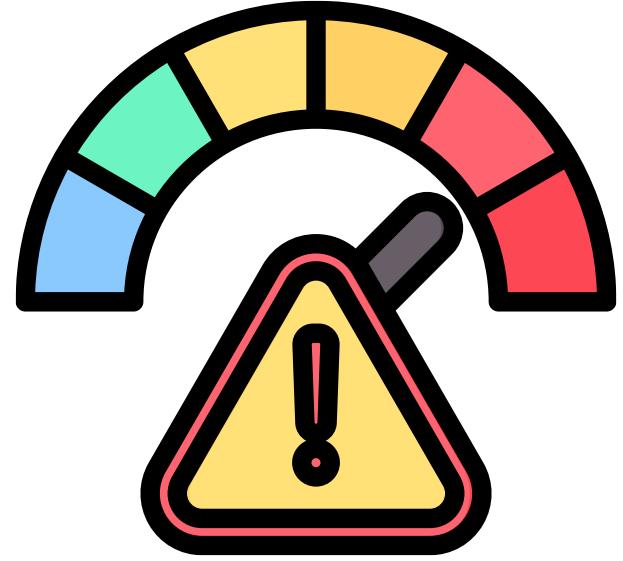
Comparativo das Técnicas



Fonte: Autoria Própria

Impacto dos Hiperparâmetros

- Taxa de Aprendizado (Learning Rate, α);
- Número de Neurônios e Camadas (Capacidade do Modelo);
- Tamanho do Batch (Batch Size).

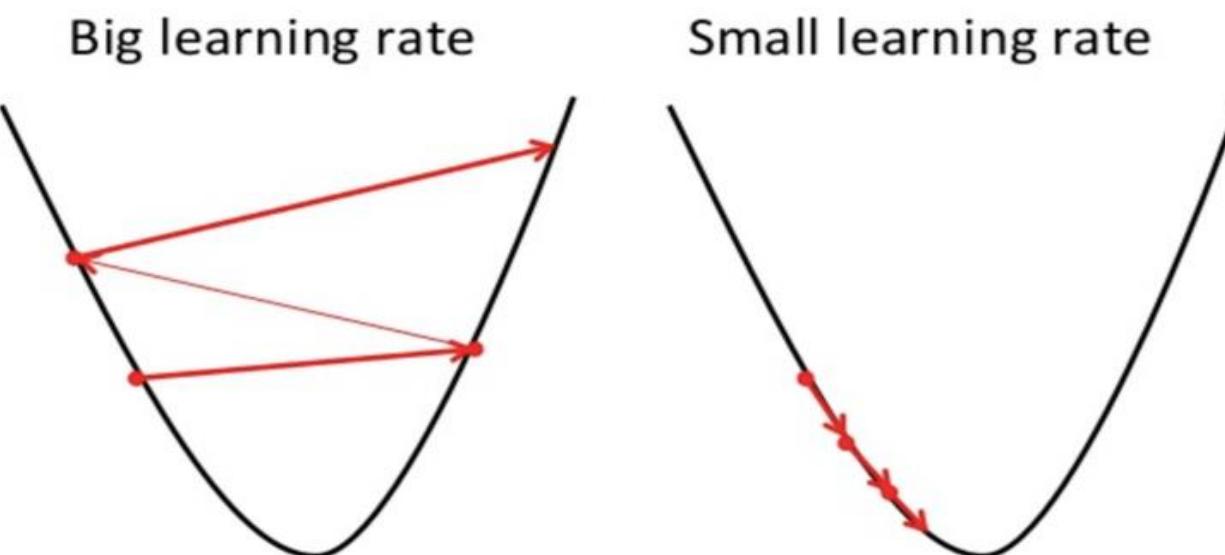


Impacto dos Hiperparâmetros

Taxa de Aprendizado (Learning Rate, α)

- O tamanho do passo que o otimizador dá na direção do gradiente negativo.
- Com a muito alta, o otimizador pode divergir (explodir) ou oscilar em torno do mínimo sem conseguir convergir;
- Com a muito baixa, a convergência extremamente lenta, risco de parar em um mínimo local ruim.

Comparação na taxa de Aprendizagem



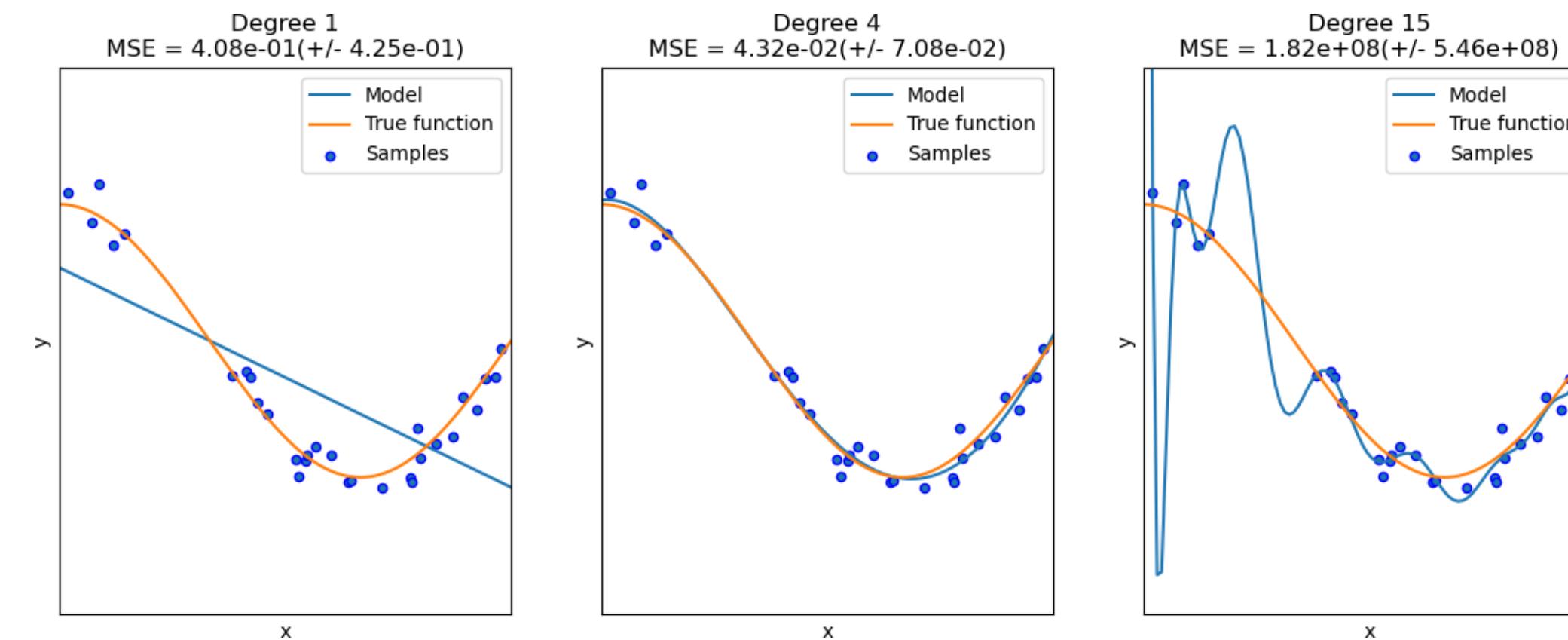
Fonte: ResearchGate

Impacto dos Hiperparâmetros

Número de Neurônios e Camadas (Capacidade do Modelo)

- Determina a complexidade e a capacidade de aprendizado da rede;
- Com Poucos neurônios/camadas (baixa capacidade), pode levar ao underfitting;
- Com Muitos neurônios/camadas (alta capacidade), pode levar ao overfitting;

Comparação de Modelos: Underfitting, Ajuste Ideal e Overfitting



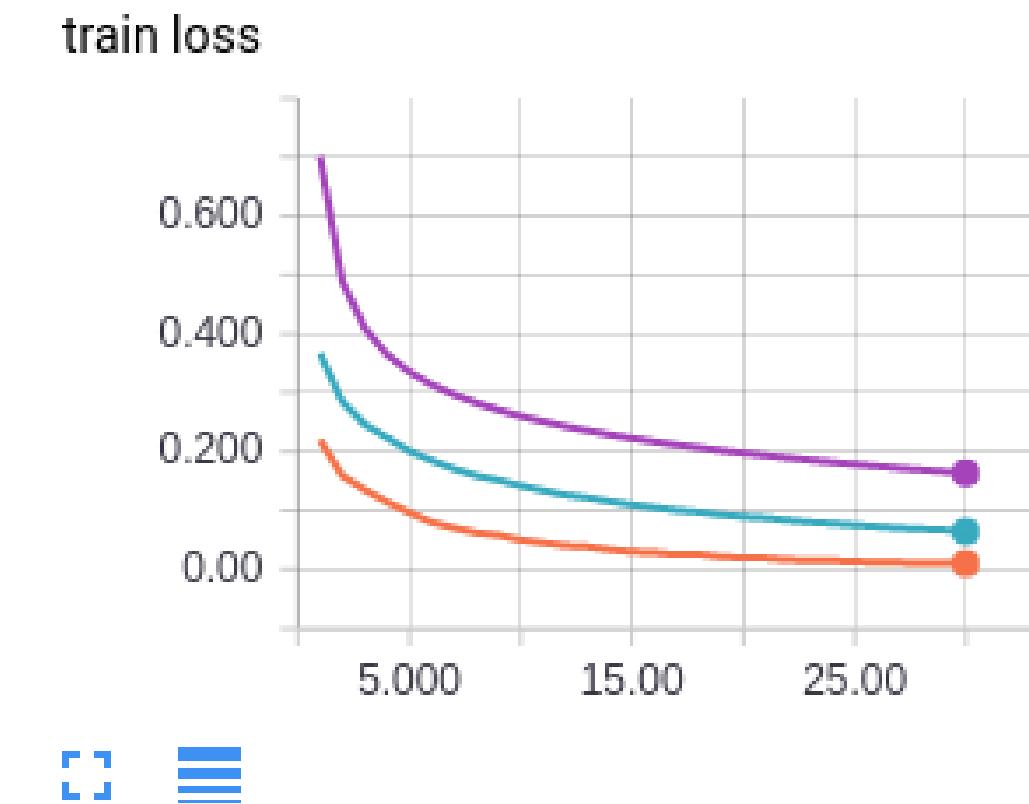
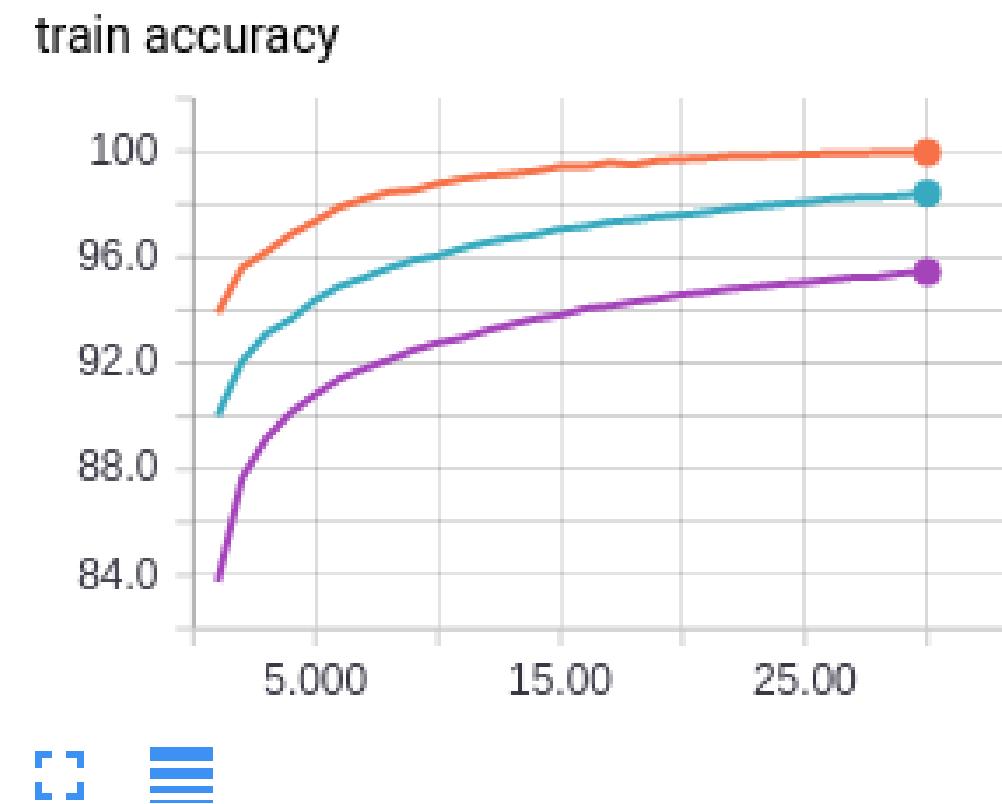
Fonte: Scikit-learn.

Impacto dos Hiperparâmetros

Tamanho do Batch (Batch Size)

- Número de exemplos de treino utilizados em uma única iteração (atualização dos pesos);
- Lotes pequenos levam a atualizações mais ruidosas, mas podem ajudar na generalização;
- Lotes grandes são computacionalmente eficientes, mas podem convergir para mínimos "mais nítidos" (pior generalização).

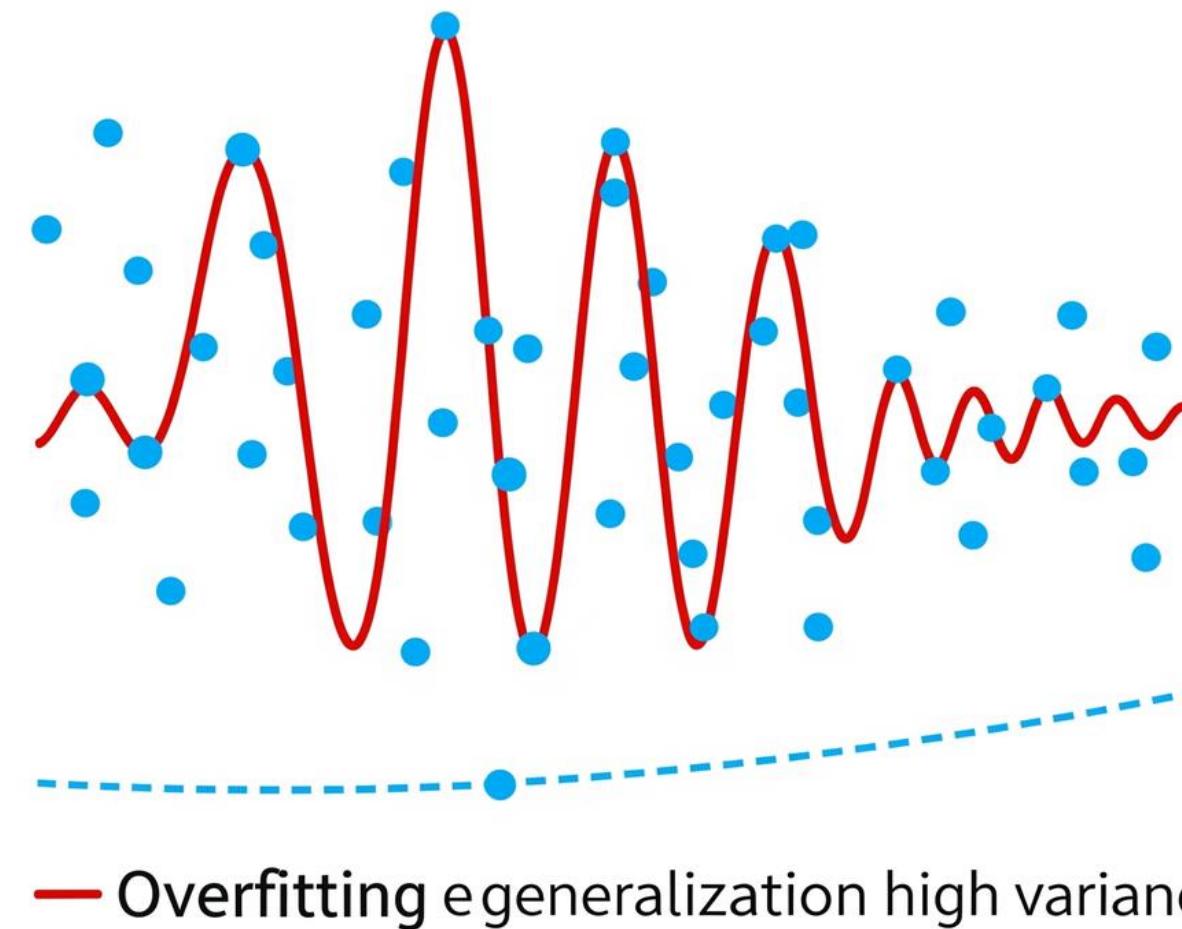
Evolução da acurácia de treino (train accuracy) e da perda de treino (train loss) ao longo das iterações.



Fonte: Deep Learning Book

Técnicas para Evitar Overfitting Hiperparâmetros

Representação de como funciona a técnica Overfitting



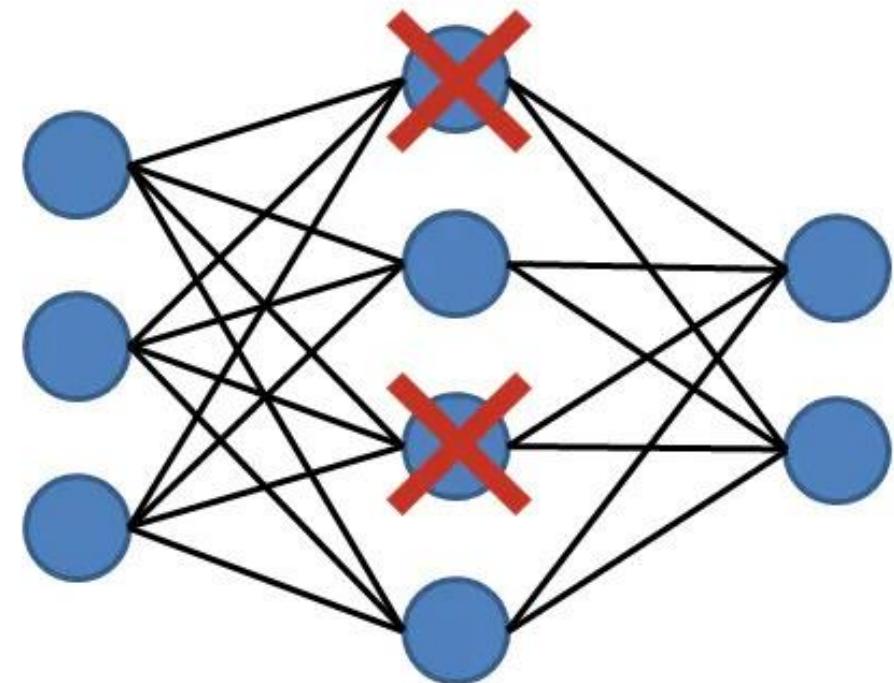
Fonte: Reaserch Gate

Técnicas para Evitar Overfitting (Hiperparâmetros)

Dropout

- Uma técnica de regularização onde a contribuição de neurônios é temporariamente removida da rede durante o treino;
- O Hiperparâmetro Principal é a Taxa de Dropout.

Representação de como funciona a técnica de Dropout

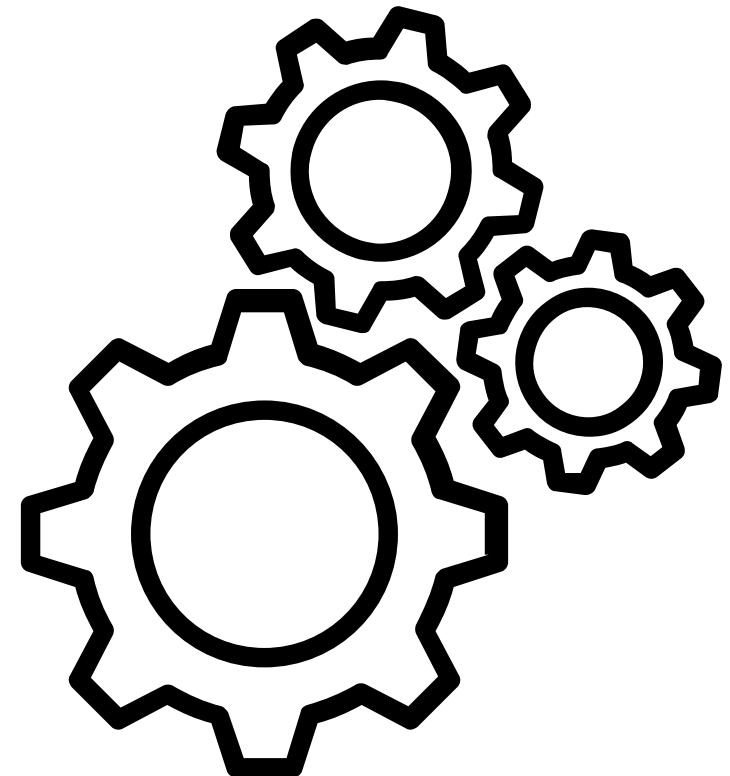


Fonte: Medium

Técnicas para Evitar Overfitting (Hiperparâmetros)

Regularização L1/L2

- Adiciona um termo de penalidade à função de perda, cuja intensidade é controlada por um hiperparâmetro;
- O Hiperparâmetro Principal é a Força da Regularização (λ , lambda).

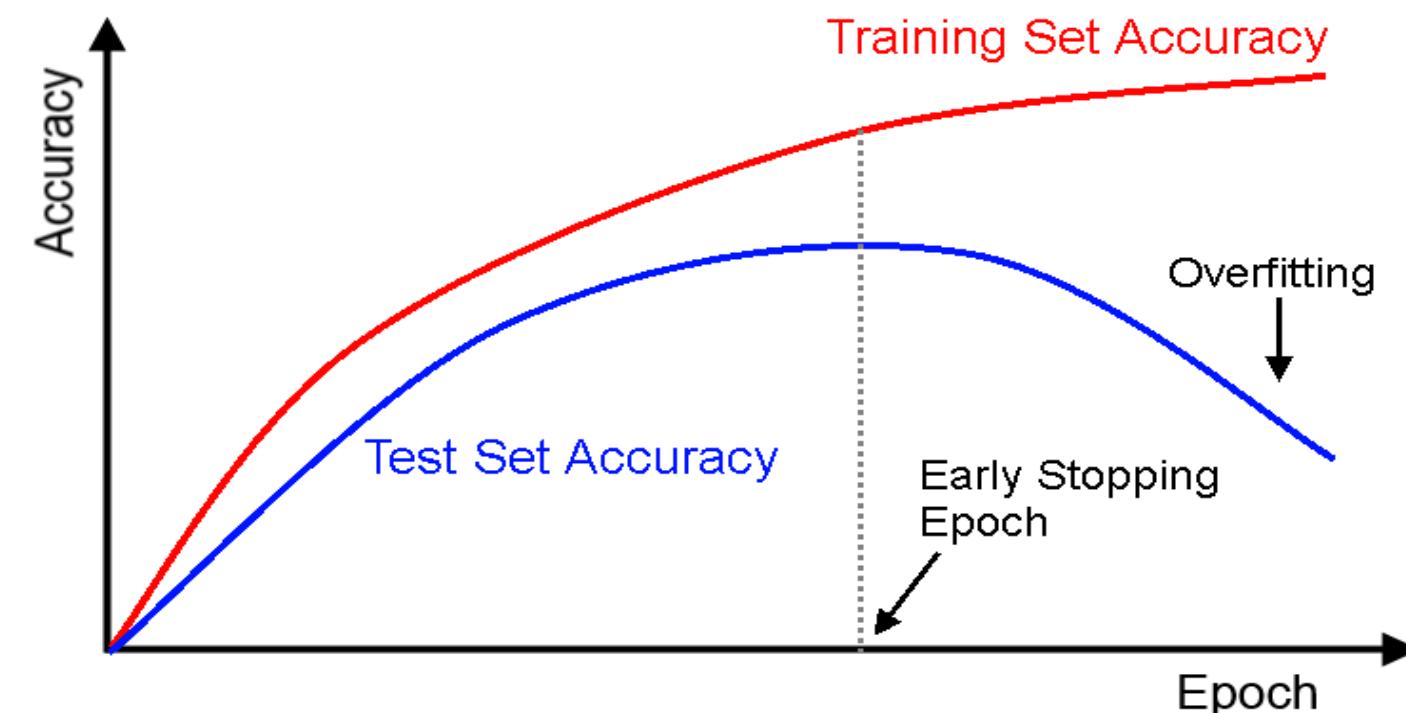


Técnicas para Evitar Overfitting (Hiperparâmetros)

Early Stopping (parada antecipada)

- Técnica que interrompe o treinamento quando a performance em um conjunto de validação para de melhorar;
- O Hiperparâmetro Principal é a "Paciência" (Patience).

Curvas de Aprendizagem da Acurácia de Treino e Teste, Ilustrando Overfitting e o Ponto de Early Stopping.



Fonte: Deep Learning Book

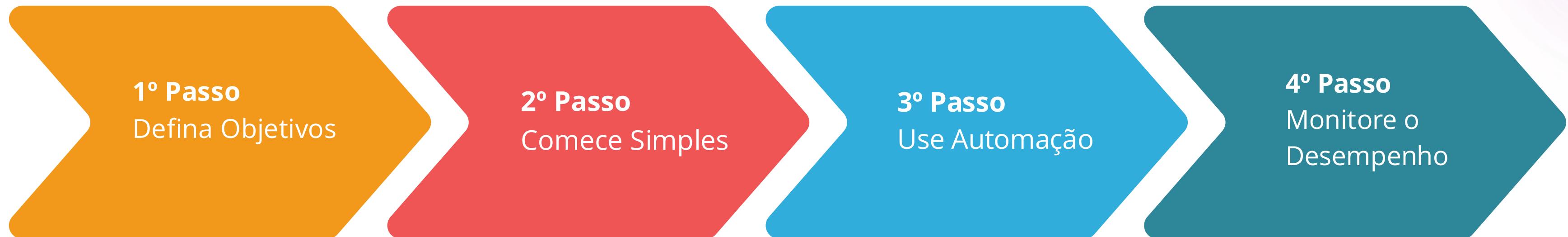
Por que Otimizar Hiperparâmetros é Importante?

- Impacto no Desempenho;
- Underfitting vs. Overfitting;
- Eficiência;
- Obtenção do Estado da Arte;

APLICAÇÕES DOS AJUSTES DE HIPERPARÂMETRO

- Classificação de Imagens;
- Processamento de Linguagem Natural (PLN);
- Análise Preditiva.

Melhores Práticas para o ajuste de Hiperparâmetros





Atividade

- **Atividade: Visualização do Desvanecimento do Gradiente.**

Referências

- HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. **Automated machine learning: methods, systems, challenges.** 1. ed. Cham: Springer, 2019. ISBN 978-3-030-05318-5.