# Retropropagação em Redes Neurais

Eduardo Adame

**Redes Neurais**

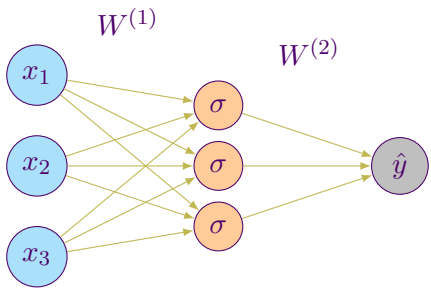
20 de agosto de 2025

# Como Treinar uma Rede Neural?

**Processo de Treinamento:**

- Inserir dados de treino
- Obter a saída da rede
- Comparar com respostas corretas
- Calcular função de perda $J$
- Ajustar pesos e repetir!

**Backpropagation:**
Usa cálculo para determinar como ajustar cada peso



# Descida do Gradiente em Redes Neurais


## Algoritmo de Treinamento

1. **Fazer predição** (forward pass)
2. **Calcular perda** $J(\mathbf{y}, \hat{\mathbf{y}})$
3. Calcular gradiente da função de perda em relação aos parâmetros
4. **Atualizar parâmetros** dando um passo na direção oposta
5. **Iterar** até convergência

## Questão Central

Como calcular $\dfrac{\partial J}{\partial W_k}$ para cada peso $W_k$ na rede?

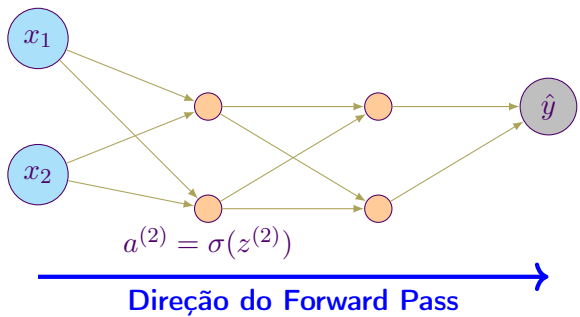
**Resposta:** Regra da Cadeia!

# Forward Propagation

$$z^{(2)} = XW^{(1)}$$

**Etapas do Forward Pass:**

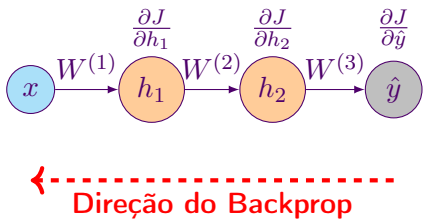
1. Inserir entrada $\mathbf{x}$
2. Calcular cada camada sequencialmente
3. Obter saída $\hat{\mathbf{y}}$
4. Avaliar perda $J(\mathbf{y}, \hat{\mathbf{y}})$



# Backpropagation: A Intuição

**O que queremos calcular:**

$$\frac{\partial J}{\partial W_k} \text{ para todo peso } W_k$$




**Estratégia:**

- Começar do erro na saída
- Propagar o erro de trás para frente
- Usar a regra da cadeia repetidamente
- Calcular gradientes camada por camada

# Cálculo dos Gradientes

## Fórmulas do Backpropagation

Para uma rede de 3 camadas com ativação sigmoid:

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{\mathbf{y}} - \mathbf{y}) \cdot a^{(3)} \tag{1}$$

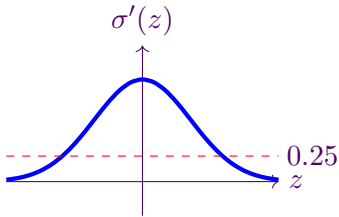
$$\frac{\partial J}{\partial W^{(2)}} = (\hat{\mathbf{y}} - \mathbf{y}) \cdot W^{(3)} \cdot \sigma'(z^{(3)}) \cdot a^{(2)} \tag{2}$$

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{\mathbf{y}} - \mathbf{y}) \cdot W^{(3)} \cdot \sigma'(z^{(3)}) \cdot W^{(2)} \cdot \sigma'(z^{(2)}) \cdot \mathbf{x} \tag{3}$$

# Problema do Gradiente Desvanecente


**O Problema:**

- $\sigma'(z) = \sigma(z)(1 - \sigma(z)) \leq 0.25$
- Gradientes ficam muito pequenos em camadas iniciais
- Multiplicação de muitos valores $< 1$
- Aprendizado extremamente lento ou para



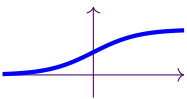

Derivada sempre $\leq 0.25$

**Solução:**

- Usar outras funções de ativação
- ReLU: $f(z) = \max(0, z)$
- Leaky ReLU, tanh, etc.

# Funções de Ativação Alternativas

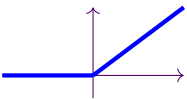

**Sigmoid**



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Gradiente desvanece

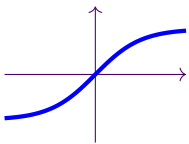
**ReLU**



$$f(z) = \max(0, z)$$

Gradiente preservado

**Tanh**



$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

Centrado em zero

# Como Escolher a Função de Ativação


**Recomendações Práticas:**

- Camadas ocultas: ReLU (default), Leaky ReLU, ELU ou SiLU
- Camada de saída:
  - Regressão: Linear ou nenhuma ativação
  - Classificação binária: Sigmoid
  - Classificação multiclasse: Softmax
  - Valores positivos: ReLU
- Experimente diferentes funções para seu problema específico
- Monitore a proporção de neurônios mortos (para ReLU)

**Regra de Ouro:**

Não existe uma função ideal para todos os casos!

# Algoritmo Completo de Backpropagation

## Pseudocódigo

**Para** cada época de treinamento:

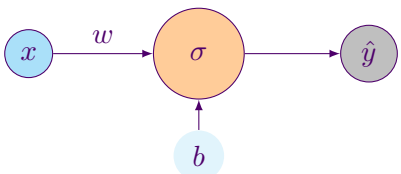
**Para** cada batch de dados:

1. **Forward Pass:**
   Calcular $a^{(l)} = \sigma(z^{(l)})$ para cada camada $l$
2. **Calcular Perda:**
   $J = \frac{1}{m} \sum_{i=1}^{m} L(y_i, \hat{y}_i)$
3. **Backward Pass:**
   Calcular $\delta^{(L)} = \nabla_a J \odot \sigma'(z^{(L)})$
   **Para** $l = L-1, L-2, ..., 1$:
   $\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(l)})$
4. **Atualizar Pesos:**
   $W^{(l)} = W^{(l)} - \alpha \cdot \delta^{(l)} \cdot (a^{(l-1)})^T$

# Exemplo Prático: Cálculo de um Gradiente

**Cenário Simplificado:**

- Rede com apenas 1 neurônio
- Entrada $x$, peso $w$, bias $b$
- $\hat{y} = \sigma(z)$ onde $z = w \cdot x + b$
- Função de perda: $J = \frac{1}{2}(y - \hat{y})^2$




$$z = w \cdot x + b$$
$$\hat{y} = \sigma(z)$$

**Cálculo do Gradiente:**

$$\begin{aligned}\frac{\partial J}{\partial w} &= \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w} \\ &= -(y - \hat{y}) \cdot \sigma'(z) \cdot x\end{aligned}$$

**Esta mesma lógica se aplica a redes complexas através da regra da cadeia!**

# Resumo e Próximos Passos


**O que aprendemos:**

- Forward propagation
- Backpropagation
- Cálculo de gradientes
- Regra da cadeia
- Problema do gradiente desvanecente
- Funções de ativação alternativas

**Próxima aula:**

- Otimizadores
- Regularização (Dropout, L2)
- Batch Normalization
- Inicialização de pesos
- Implementação prática

Backpropagation é o coração do aprendizado profundo!

# Obrigado!

Alguma dúvida?

Agora vamos implementar backpropagation!