Notas de Aula - Semana 3

Retropropagação em Redes Neurais

Curso de Redes Neurais

Eduardo Adame

27 de agosto de 2025

Sumário

1 Introdução

O algoritmo de retropropagação (*backpropagation*) é o método fundamental para treinar redes neurais artificiais. Desenvolvido na década de 1980, este algoritmo revolucionou o campo de aprendizado de máquina ao permitir o treinamento eficiente de redes neurais profundas. Nesta aula, exploraremos os fundamentos matemáticos e a implementação prática deste algoritmo essencial.

2 Como Treinar uma Rede Neural

2.1 O Problema do Treinamento

O treinamento de uma rede neural consiste em ajustar seus parâmetros (pesos e bias) para minimizar uma função de perda que mede a discrepância entre as predições da rede e os valores desejados. Este processo envolve:

1. **Forward Pass**: Calcular a saída da rede para uma entrada dada
2. **Cálculo da Perda**: Medir o erro entre a saída e o valor esperado
3. **Backward Pass**: Calcular gradientes da perda em relação aos parâmetros
4. **Atualização**: Ajustar os parâmetros usando os gradientes

2.2 A Necessidade do Backpropagation

Observação 2.1: Desafio Computacional

Em uma rede neural com milhões de parâmetros, calcular gradientes por diferenças finitas seria computacionalmente proibitivo. O backpropagation resolve este problema usando a regra da cadeia do cálculo para calcular todos os gradientes em uma única passada pela rede.

3 Descida do Gradiente em Redes Neurais

3.1 Algoritmo de Treinamento

O processo de treinamento de uma rede neural segue o algoritmo de descida do gradiente:

Algoritmo 3.1: Descida do Gradiente para Redes Neurais

1: Inicializar pesos $W^{(l)}$ e bias $b^{(l)}$ para todas as camadas l
2: **repeat**
3: **Forward Pass:** Calcular saída $\hat{\mathbf{y}}$ para entrada $\mathbf{x}$
4: **Calcular Perda:** $J = L(\mathbf{y}, \hat{\mathbf{y}})$
5: **Backward Pass:** Calcular $\frac{\partial J}{\partial W^{(l)}}$ e $\frac{\partial J}{\partial b^{(l)}}$ para todo l
6: **Atualizar Parâmetros:**

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial J}{\partial W^{(l)}} \tag{1}$$

$$b^{(l)} \leftarrow b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}} \tag{2}$$

7: **until** convergência

3.2 A Questão Central

A questão central que o backpropagation resolve é:

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

A resposta está na aplicação sistemática da **regra da cadeia** do cálculo.

4 Forward Propagation

4.1 Computação Direta

O forward pass calcula a saída da rede camada por camada:

Definição 4.1: Forward Propagation

Para uma rede com L camadas, o forward pass computa:

$$a^{(0)} = \mathbf{x} \quad \text{(entrada)} \tag{3}$$

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \quad \text{para } l = 1, ..., L \tag{4}$$

$$a^{(l)} = f^{(l)}(z^{(l)}) \quad \text{para } l = 1, ..., L \tag{5}$$

$$\hat{\mathbf{y}} = a^{(L)} \quad \text{(saída)} \tag{6}$$

onde $f^{(l)}$ é a função de ativação da camada l.

4.2 Exemplo de Forward Pass

Exemplo 4.1: Forward Pass em Rede de 2 Camadas

Considere uma rede com arquitetura [2, 3, 1] (2 entradas, 3 neurônios ocultos, 1 saída).
Dado $\mathbf{x} = [0.5, -0.2]^T$, e matrizes de pesos:

$$W^{(1)} = \begin{bmatrix} 1 & -1 \\ 0.5 & 2 \\ -0.3 & 0.8 \end{bmatrix}, \quad W^{(2)} = \begin{bmatrix} 0.7 & -0.5 & 1.2 \end{bmatrix}$$

Com bias $b^{(1)} = [0.1, -0.2, 0.3]^T$ e $b^{(2)} = [0.15]$:
Camada 1:

$$z^{(1)} = W^{(1)}\mathbf{x} + b^{(1)} = \begin{bmatrix} 0.8 \\ -0.25 \\ -0.01 \end{bmatrix} \tag{7}$$

$$a^{(1)} = \sigma(z^{(1)}) = \begin{bmatrix} 0.690 \\ 0.438 \\ 0.498 \end{bmatrix} \tag{8}$$

Camada 2:

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)} = [0.561] \tag{9}$$

$$\hat{y} = \sigma(z^{(2)}) = 0.637 \tag{10}$$

5 Backpropagation: A Intuição

5.1 Propagação do Erro

O backpropagation funciona propagando o erro da saída para as camadas anteriores. A intuição é:

1. Calcular o erro na saída
2. Determinar quanto cada neurônio da última camada contribuiu para este erro
3. Propagar esta informação para camadas anteriores
4. Repetir até chegar na entrada

5.2 Fundamento Matemático: Regra da Cadeia

Teorema 5.1: Regra da Cadeia para Backpropagation

Para uma função composta $J = L(f_L(f_{L-1}(...f_1(\mathbf{x}))))$, o gradiente em relação aos parâmetros da camada l é:

$$\frac{\partial J}{\partial W^{(l)}} = \frac{\partial J}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial W^{(l)}}$$

onde $\frac{\partial J}{\partial z^{(l)}}$ é o "erro"propagado até a camada l.

6 Cálculo dos Gradientes

6.1 Notação Delta

Para simplificar a notação, definimos o erro da camada l como:

$$\delta^{(l)} = \frac{\partial J}{\partial z^{(l)}}$$

Este termo representa o quanto a entrada líquida $z^{(l)}$ contribui para o erro total.

6.2 Equações do Backpropagation

Definição 6.1: Equações Fundamentais do Backpropagation

Para uma rede neural feedforward:
Erro da última camada:

$$\delta^{(L)} = \nabla_a J \odot f'(z^{(L)})$$

Erro das camadas anteriores:

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

Gradientes dos pesos:

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

Gradientes dos bias:

$$\frac{\partial J}{\partial b^{(l)}} = \delta^{(l)}$$

onde $\odot$ denota o produto de Hadamard (elemento por elemento).

6.3 Exemplo Detalhado

Exemplo 6.1: Cálculo de Gradientes

Continuando o exemplo anterior, suponha que o valor esperado seja $y = 1$ e usamos MSE como função de perda:

$$J = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(1 - 0.637)^2 = 0.0659$$

Passo 1: Erro da última camada

$$\begin{aligned} \delta^{(2)} &= (0.637 - 1) \cdot \sigma'(0.561) && (11) \\ &= -0.363 \cdot 0.637 \cdot (1 - 0.637) && (12) \\ &= -0.084 && (13) \end{aligned}$$

Passo 2: Propagar para camada anterior

$$\begin{aligned} \delta^{(1)} &= (W^{(2)})^T \delta^{(2)} \odot \sigma'(z^{(1)}) && (14) \\ &= \begin{bmatrix} 0.7 \\ -0.5 \\ 1.2 \end{bmatrix} \cdot (-0.084) \odot \begin{bmatrix} 0.214 \\ 0.246 \\ 0.250 \end{bmatrix} && (15) \\ &= \begin{bmatrix} -0.0126 \\ 0.0103 \\ -0.0252 \end{bmatrix} && (16) \end{aligned}$$

Passo 3: Calcular gradientes

$$\begin{aligned} \frac{\partial J}{\partial W^{(2)}} &= \delta^{(2)} \cdot (a^{(1)})^T = -0.084 \cdot [0.690, 0.438, 0.498] && (17) \\ &= [-0.058, -0.037, -0.042] && (18) \end{aligned}$$

7 Problema do Gradiente Desvanecente

7.1 Descrição do Problema

Um dos principais desafios no treinamento de redes neurais profundas é o problema do gradiente desvanecente.

Observação 7.1: Gradiente Desvanecente

Quando usamos a função sigmoid, sua derivada satisfaz:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \leq 0.25$$

Em redes profundas, os gradientes são produtos de muitas derivadas:

$$\delta^{(1)} = \delta^{(L)} \cdot \prod_{l=2}^{L} W^{(l)} \cdot f'(z^{(l-1)})$$

Se cada $f'(z^{(l)}) \leq 0.25$, o gradiente decai exponencialmente com a profundidade.

7.2 Análise Matemática

Para uma rede com L camadas usando sigmoid:

$$|\delta^{(1)}| \leq |\delta^{(L)}| \cdot \prod_{l=2}^{L} ||W^{(l)}|| \cdot 0.25$$

Se $||W^{(l)}|| \approx 1$, então:

$$|\delta^{(1)}| \leq |\delta^{(L)}| \cdot (0.25)^{L-1}$$

Para $L = 10$ camadas:

$$|\delta^{(1)}| \leq |\delta^{(10)}| \cdot (0.25)^{9} \approx |\delta^{(10)}| \cdot 3.8 \times 10^{-6}$$

7.3 Consequências Práticas

1. **Aprendizado lento**: Camadas iniciais aprendem muito lentamente
2. **Estagnação**: Pesos podem parar de atualizar completamente
3. **Degradação do desempenho**: Redes muito profundas podem ter desempenho pior que redes rasas

8 Funções de Ativação Alternativas

8.1 ReLU (Rectified Linear Unit)

Definição 8.1: Função ReLU

A função ReLU e sua derivada são definidas como:

$$\text{ReLU}(z) = \max(0, z) \tag{19}$$

$$\text{ReLU}'(z) = \begin{cases} 1, & \text{se } z > 0 \\ 0, & \text{se } z < 0 \\ \text{indefinido}, & \text{se } z = 0 \end{cases} \tag{20}$$

Vantagens da ReLU:

- Derivada constante (0 ou 1) - não há atenuação do gradiente
- Computacionalmente eficiente
- Promove esparsidade na rede
- Convergência mais rápida

Desvantagens:

- "Dying ReLU": neurônios podem ficar permanentemente inativos
- Não é diferenciável em $z = 0$
- Saída não limitada superiormente

8.2 Função Tanh

Definição 8.2: Função Tangente Hiperbólica

$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1} = \frac{2}{1 + e^{-2z}} - 1 \tag{21}$$

$$\tanh'(z) = 1 - \tanh^2(z) \tag{22}$$

Vantagens da Tanh:

- Centrada em zero (saída entre -1 e 1)
- Derivada máxima de 1 (melhor que sigmoid)
- Gradientes mais fortes que sigmoid

8.3 Leaky ReLU e Variantes

Definição 8.3: Leaky ReLU

$$\text{LeakyReLU}(z) = \begin{cases} z, & \text{se } z > 0 \\ \alpha z, & \text{se } z \leq 0 \end{cases} \tag{23}$$

$$\text{LeakyReLU}'(z) = \begin{cases} 1, & \text{se } z > 0 \\ \alpha, & \text{se } z \leq 0 \end{cases} \tag{24}$$

onde α é um pequeno valor positivo (tipicamente 0.01).