Neural Network from Scratch

Implementação e Aplicação

Felipe Adeildo da Silva

Novo Ensino Suplementar

12 de Julho de 2024

Conteúdo

1. Introdução
2. O que é uma Rede Neural?
3. Funções de Ativação
4. Forward Propagation
5. Função de Custo
6. Backpropagation e Descida do Gradiente
7. Implementação Prática
8. Conclusão

Introdução

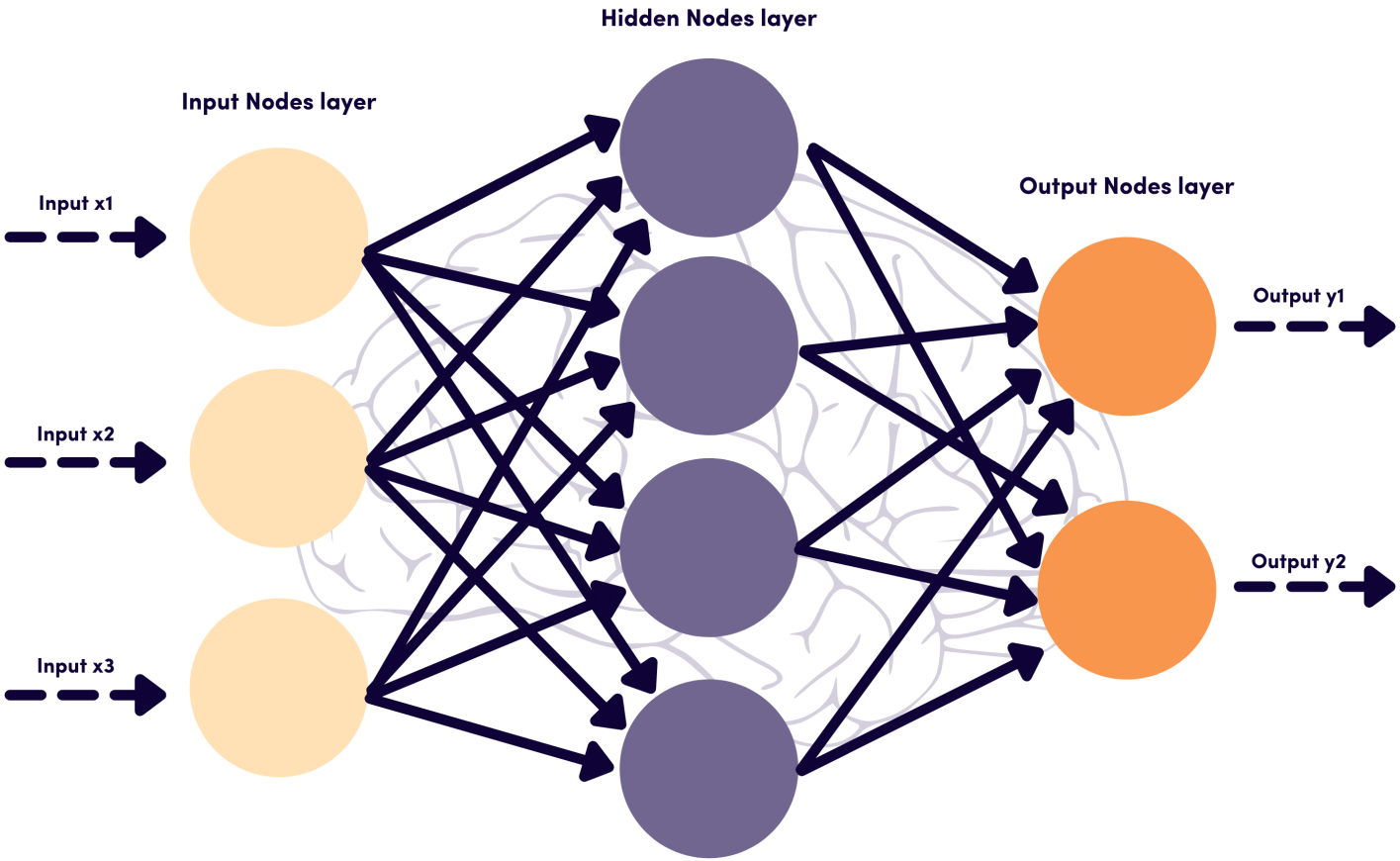
Introdução

- **Apresentação do Projeto:** "Neural Network from Scratch"
- **Objetivo:** Compreender o funcionamento interno de uma rede neural antes de utilizar bibliotecas de alto nível, como o TensorFlow.
- **Metodologia:** Abordagem matemática para entender como uma rede neural aprende padrões, minimiza a função de custo e opera através de suas camadas.

O que é uma Rede Neural?

O que é uma Rede Neural?

- Modelo computacional inspirado no cérebro humano.
- Composto por camadas de neurônios interconectados.
- Capacidade de aprender padrões complexos a partir dos dados.


Hidden Nodes layer
Input Nodes layer
Output Nodes layer
Input x1
Input x2
Input x3
Output y1
Output y2

Neurônios

- Neurônios são as unidades fundamentais de uma rede neural.
- Cada neurônio recebe múltiplas entradas, aplica pesos a essas entradas, soma os resultados e adiciona um viés.
- Essa soma é então passada por uma função de ativação para produzir a saída do neurônio.

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$a = \sigma(z)$$

Função de Ativação

A função $\sigma(z)$ é chamada de função de ativação e será discutida mais detalhadamente em seções posteriores.

Exemplo da Ação de um Neurônio

Example

Suponha que um neurônio receba três entradas $x_1 = 0.5$, $x_2 = 0.3$, $x_3 = 0.9$, com pesos $w_1 = 0.4$, $w_2 = 0.7$, $w_3 = 0.2$, e viés $b = 0.1$. O somatório seria:

$$z = (0.4 \cdot 0.5) + (0.7 \cdot 0.3) + (0.2 \cdot 0.9) + 0.1 = 0.65$$

A saída do neurônio após a função de ativação $\sigma(z)$ seria $a = \sigma(0.65)$.

Camadas

- As camadas são coleções de neurônios organizadas sequencialmente.
- Tipos de camadas:
  - Camada de entrada: Recebe os dados brutos.
  - Camadas ocultas: Realizam a maior parte do processamento intermediário.
  - Camada de saída: Produz a previsão ou classificação final.
- Cada camada transforma as saídas da camada anterior através de pesos, vieses e funções de ativação.

Matriz de Pesos

- Pesos determinam a força das conexões entre os neurônios.
- Uma matriz de pesos conecta a saída de uma camada à entrada da próxima camada.
- Cada neurônio em uma camada está conectado a todos os neurônios na camada seguinte.

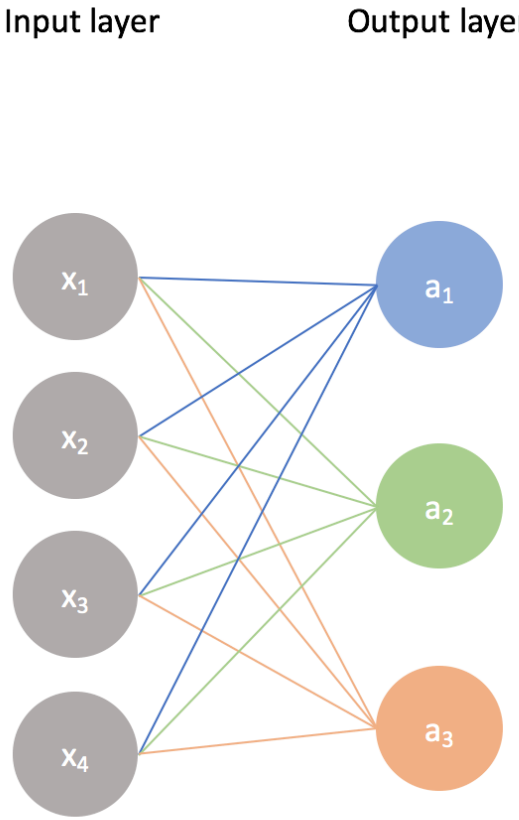
$$\mathrm{Z}^{(l)} = \mathrm{W}^{(l)}\mathrm{A}^{(l-1)} + \mathrm{b}^{(l)}$$

Matriz de Pesos

Suponha que a camada $(l-1)$ tenha n neurônios e a camada l tenha m neurônios. A matriz de pesos $\mathrm{W}^{(l)}$ será de dimensão $m \times n$, onde cada elemento w_{ij} representa o peso da conexão entre o neurônio j da camada $(l-1)$ e o neurônio i da camada l.

$$\mathrm{W}^{(l)} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$

Ilustração da Atuação da Matriz de Pesos



A simple neural network

$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \xrightarrow[activation]{} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Exemplo Prático de Matriz de Pesos

Example

Suponha que tenhamos uma camada oculta com 3 neurônios e uma camada de saída com 2 neurônios. A matriz de pesos conectando essas camadas seria:

$$\mathrm{W}^{(2)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

Cada elemento w_{ij} representa o peso da conexão do neurônio i da camada oculta ao neurônio j da camada de saída.

Funções de Ativação

Funções de Ativação: Necessidade

- Introdução de não-linearidade: Permite o aprendizado de padrões complexos nos dados.
- Nossos dados geralmente são não lineares; sem funções de ativação, uma rede neural seria simplesmente um modelo de regressão linear.
- Exemplos de problemas não lineares:
  - Reconhecimento de imagens
  - Processamento de linguagem natural
  - Previsão de séries temporais

Funções de Ativação: Principais Tipos

- Principais Funções de Ativação:
  - Sigmoid
  - ReLU
  - Softmax

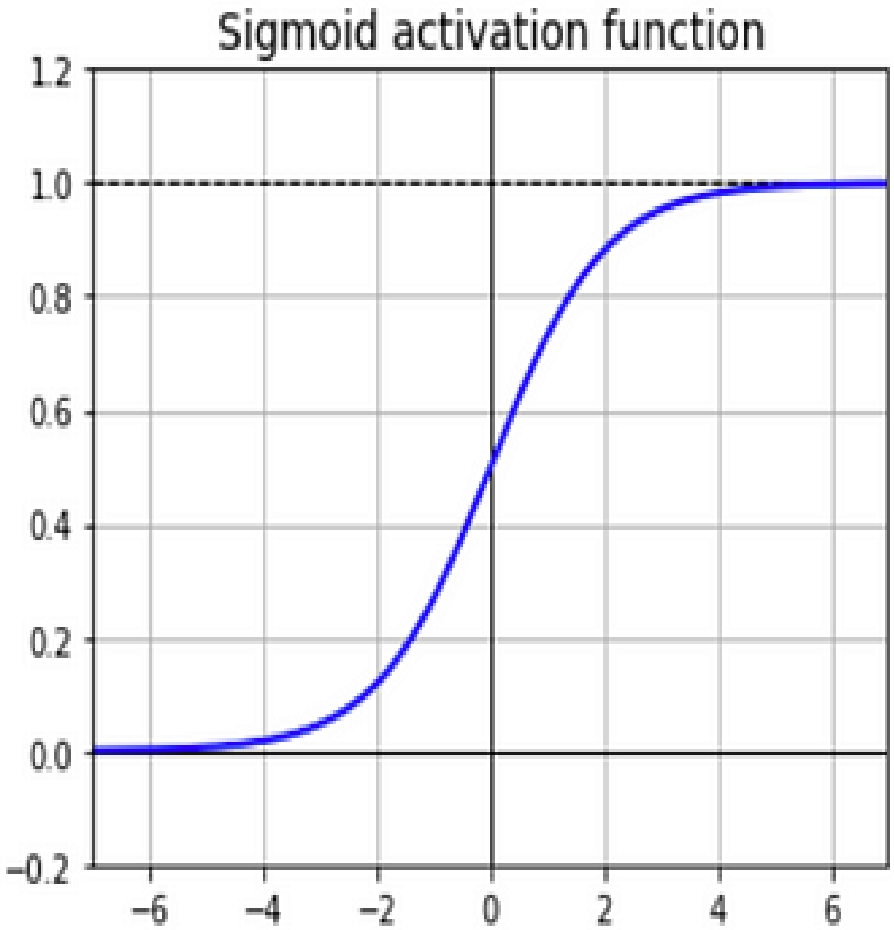
Função de Ativação: Sigmoid

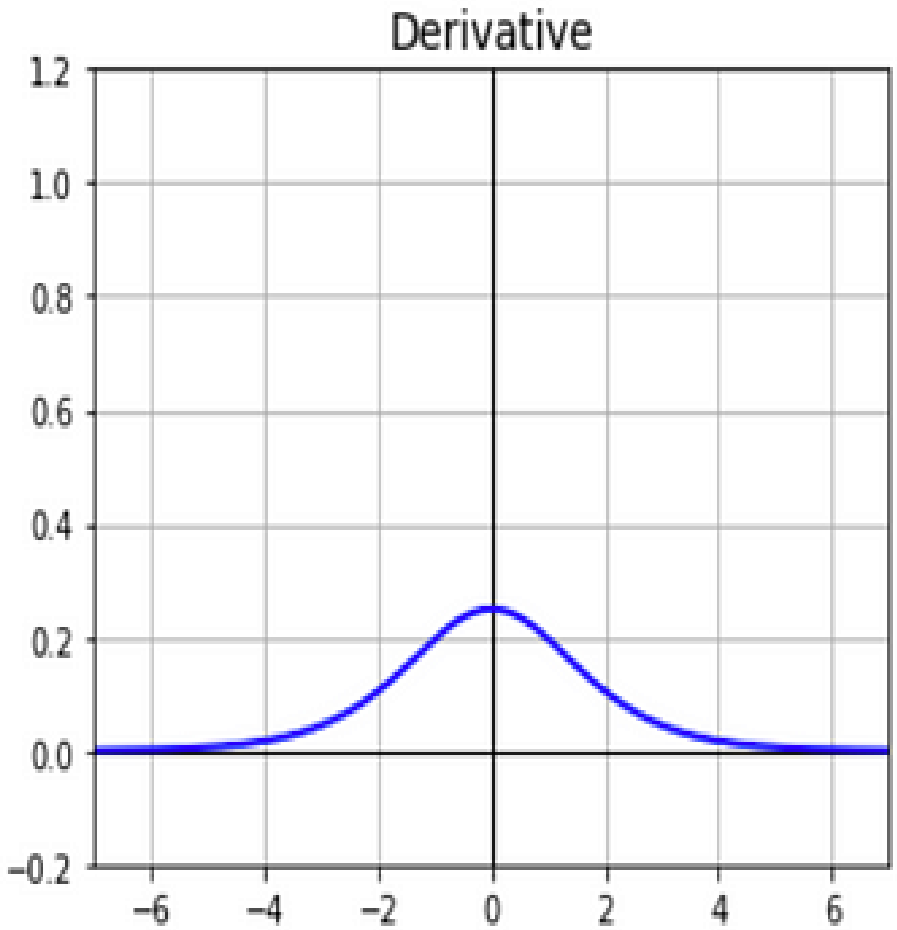
- A função sigmoid modela a saída como uma probabilidade entre 0 e 1.
- Tipo de Problema: Utilizada em problemas de classificação binária.
- Prós:
  - Saída limitada entre 0 e 1, útil para probabilidades.
- Contras:
  - Problema de gradiente desaparecendo para valores extremos.

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

Gráfico: Função de Ativação Sigmoid

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





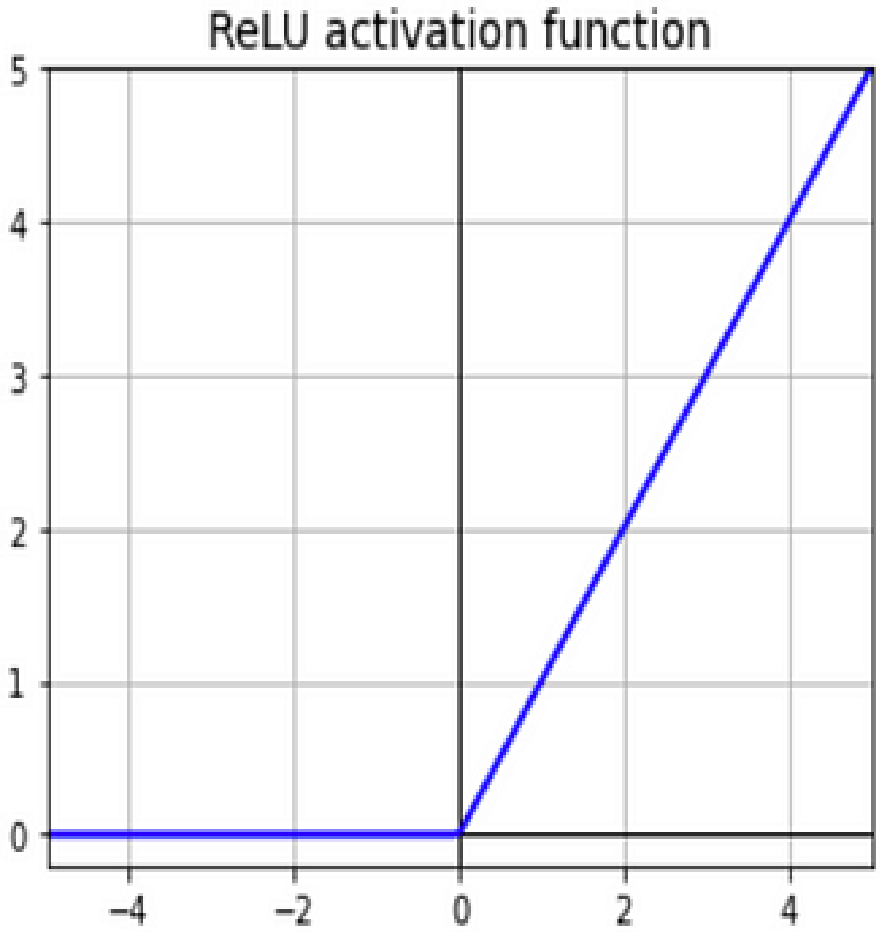
Função de Ativação: ReLU

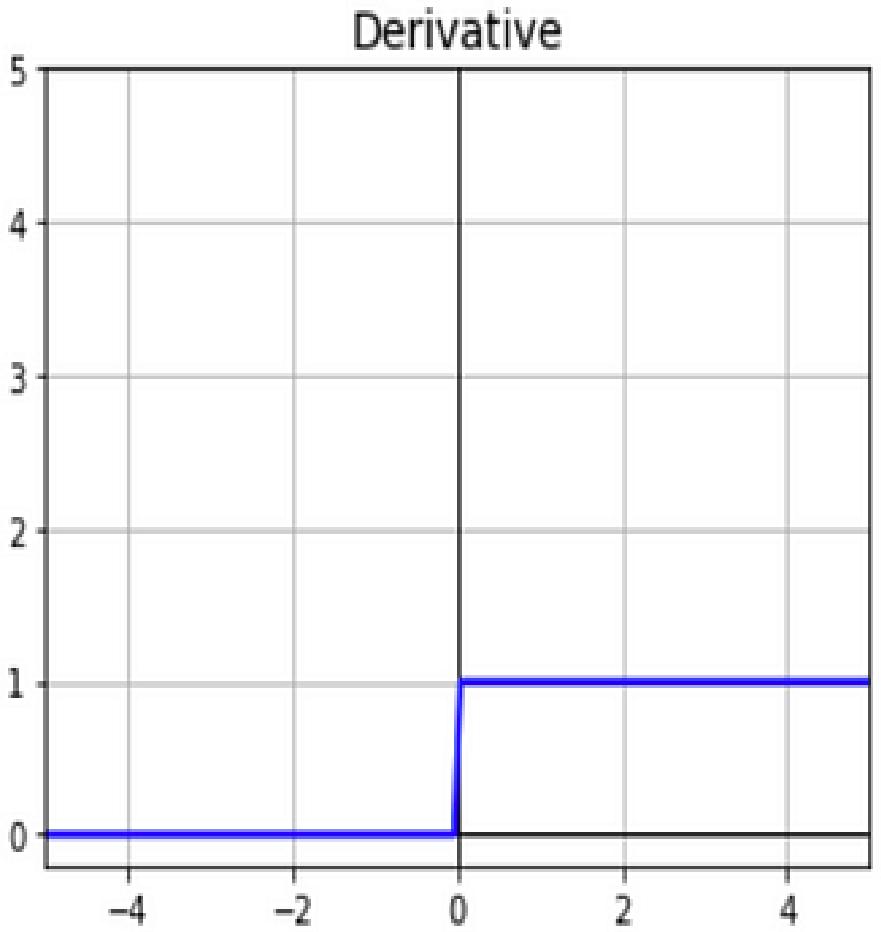
- A função ReLU (Rectified Linear Unit) permite a passagem de valores positivos diretamente.
- Tipo de Problema: Utilizada em várias arquiteturas de rede, especialmente em redes profundas.
- Prós:
  - Resolve o problema do gradiente desaparecendo.
  - Computacionalmente eficiente.
- Contras:
  - Problema de neurônios mortos (Dead Neurons).

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

Gráfico: Função de Ativação ReLU

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





Função de Ativação: Softmax

- A função Softmax transforma um vetor de valores em uma distribuição de probabilidade.
- Tipo de Problema: Utilizada em problemas de classificação multiclasse.
- Prós:
  - Fornece uma distribuição de probabilidade sobre classes.
- Contras:
  - Mais computacionalmente intensiva que outras funções de ativação.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

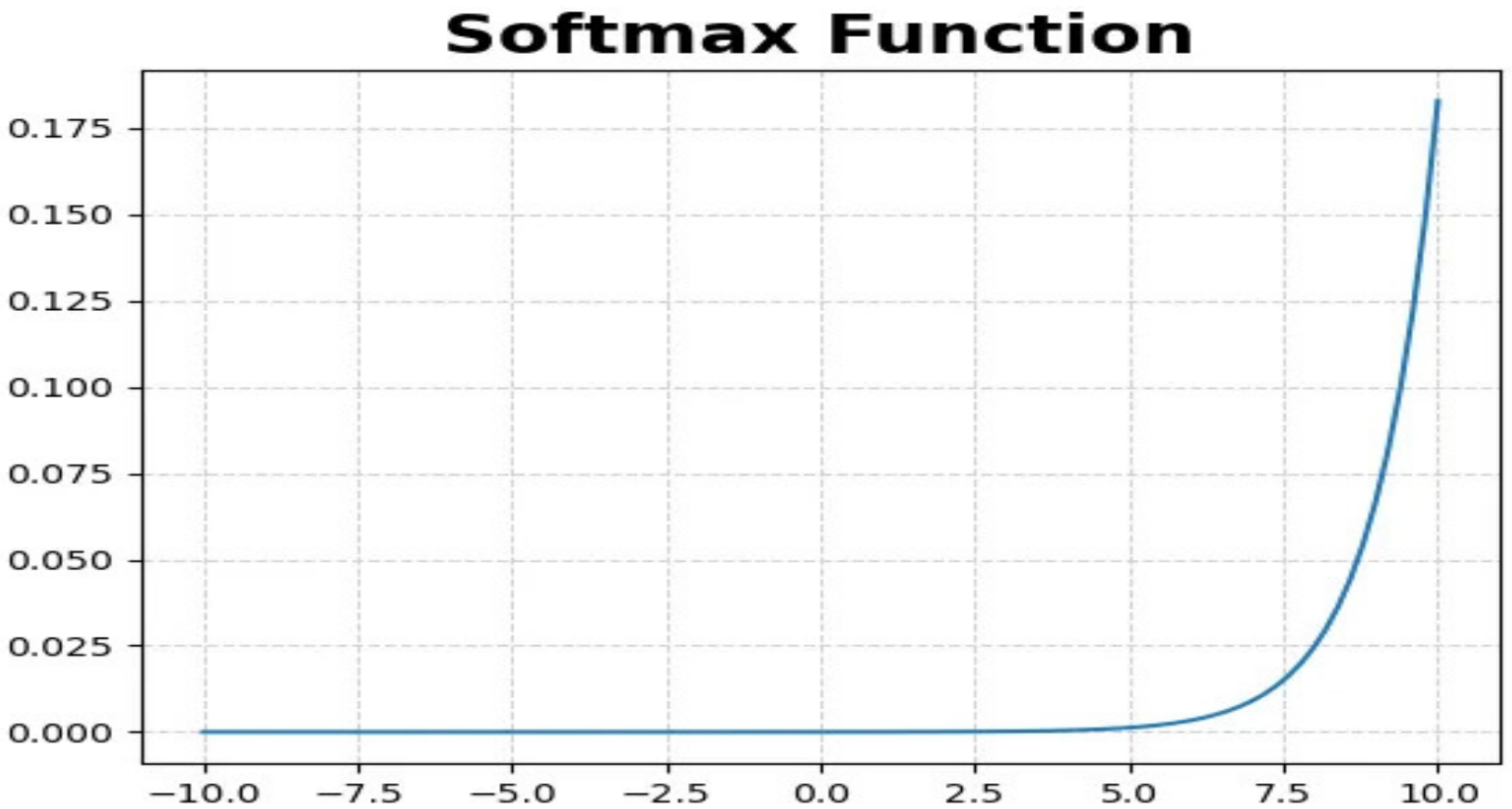
Derivada do Softmax

A função Softmax não possui uma derivada simples, mas é necessário calcular a taxa de variação para a estabilidade numérica durante a retropropagação. A derivada do Softmax é computada usando:

$$\frac{\partial \sigma(z_i)}{\partial z_j} = \sigma(z_i)(1 - \sigma(z_j))$$

Gráfico: Função de Ativação Softmax

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$



Forward Propagation

Forward Propagation

- **Objetivo do Forward Propagation**
  - Forward Propagation, também chamado de Feedforward, é o processo pelo qual uma rede neural faz previsões com base em seus pesos e vieses atuais.
  - A rede neural processa os dados de entrada através das suas camadas e gera uma saída ou predição.

Processo de Forward Propagation

- **Passagem dos Dados pela Rede**
  - Os dados de entrada são passados através da rede, camada por camada.
  - Em cada camada, os dados são transformados pelos pesos, vieses e funções de ativação.
- **Transformação Camada por Camada**
  - Para cada camada l, o processo é:
  - $$\mathrm{Z}^{(l)} = \mathrm{W}^{(l)}\mathrm{A}^{(l-1)} + \mathrm{b}^{(l)}$$
  - $$\mathrm{A}^{(l)} = \sigma(\mathrm{Z}^{(l)})$$

Exemplo Matemático

- **Equação Geral da Propagação Direta**
  - Dado um conjunto de entradas X, a primeira camada calcula:

$$\mathrm{Z}^{(1)} = \mathrm{W}^{(1)}\mathrm{X} + \mathrm{b}^{(1)}$$

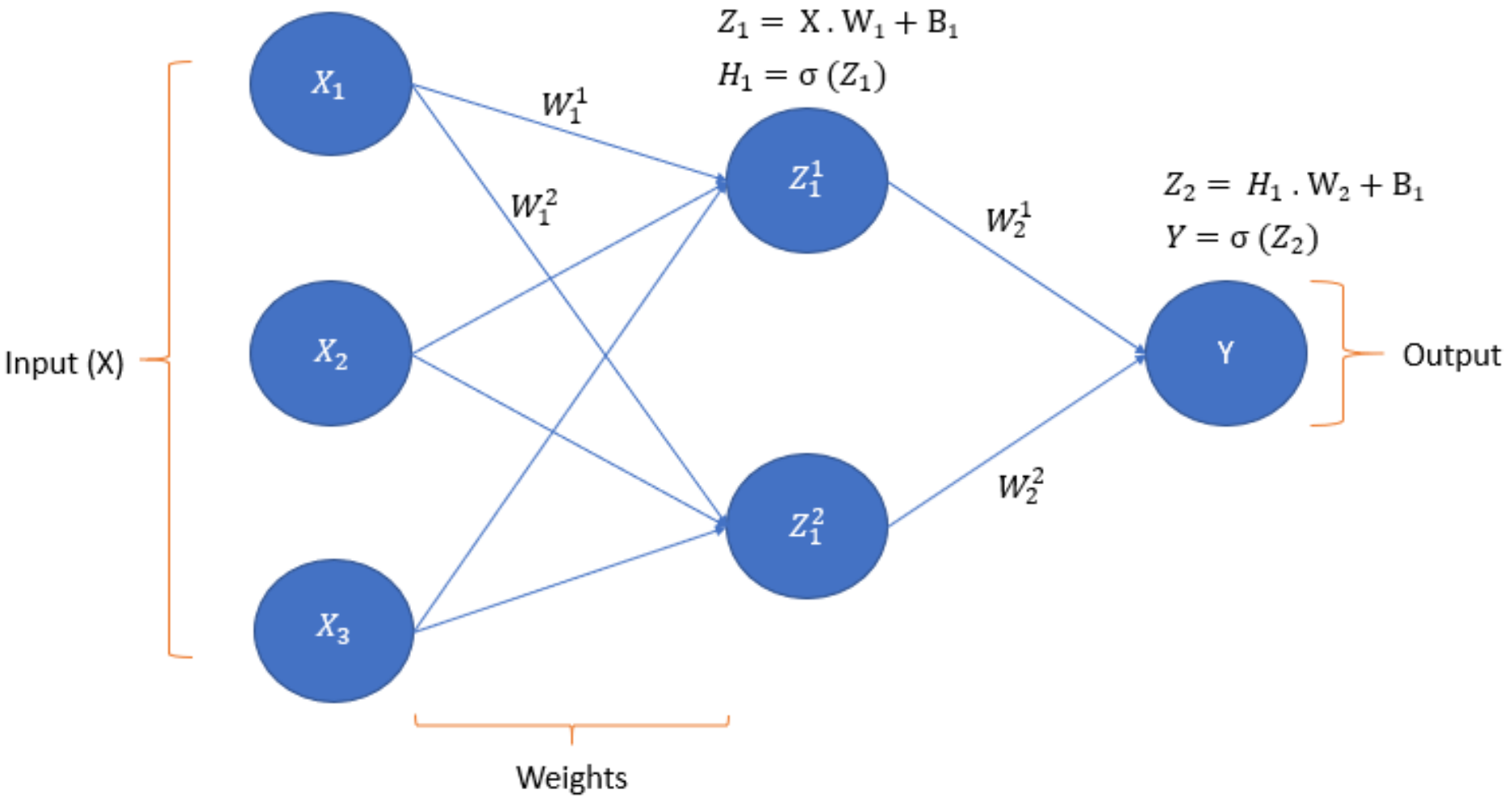
  - A ativação é aplicada:

$$\mathrm{A}^{(1)} = \sigma(\mathrm{Z}^{(1)})$$

  - Esse processo se repete para cada camada subsequente até a camada de saída.

Ilustração do Processo

- **Diagrama de Forward Propagation**



Função de Custo

Função de Custo

- **Necessidade de uma Função de Custo**
  - A função de custo mede a diferença entre as previsões da rede neural e os valores reais.
  - Ela guia o processo de treinamento ajustando os pesos para minimizar essa diferença.
- **Principais Funções de Custo**
  - Mean Squared Error (MSE)
  - Cross-Entropy Loss

Mean Squared Error (MSE)

- **O que é MSE?**
  - O MSE mede a média dos quadrados das diferenças entre as previsões e os valores reais.
- **Interpretação Matemática**
  - Fórmula:

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

  - Onde y_i são os valores reais e $\hat{y}_i$ são as previsões.
- **Prós e Contras**
  - **Prós:** Simples de implementar e entender. Sensível a grandes erros.
  - **Contras:** Sensível a outliers, que podem dominar a função de custo.
- **Quando Usar?**
  - Usado principalmente em problemas de regressão onde se quer minimizar grandes erros.

Cross-Entropy Loss

- **O que é Cross-Entropy Loss?**
  - A Cross-Entropy Loss mede a diferença entre duas distribuições de probabilidade: a distribuição real e a distribuição prevista.
- **Interpretação Matemática**
  - Fórmula:

$$\text{Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}\log(\hat{y}_{ij})$$

  - Onde:
    - n é o número de amostras.
    - c é o número de classes.
    - y_{ij} é o valor real (one-hot encoded) da i-ésima amostra para a j-ésima classe.
    - $\hat{y}_{ij}$ é a previsão de probabilidade da i-ésima amostra para a j-ésima classe.
- **Prós e Contras**
  - **Prós:** Adequada para problemas de classificação. Penaliza fortemente previsões incorretas.
  - **Contras:** Pode ser mais complexa de calcular e interpretar.
- **Quando Usar?**
  - Usada principalmente em problemas de classificação, especialmente para classificação multiclasse.

Função de Custo: MSE e Cross-Entropy Loss

Mean Squared Error (MSE)

O MSE mede a média dos quadrados das diferenças entre as previsões e os valores reais:

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

Cross-Entropy Loss

A Cross-Entropy Loss mede a diferença entre a distribuição de probabilidade real e a prevista:

$$\text{Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}\log(\hat{y}_{ij})$$

Backpropagation e Descida do Gradiente

Backpropagation

- **O que é Backpropagation?**
  - É um algoritmo usado para calcular os gradientes da função de custo em relação aos pesos da rede neural.
  - Funciona propagando o erro de volta através da rede, camada por camada.
- **Intuição por Trás do Backpropagation**
  - Calcula o erro na camada de saída.
  - Propaga esse erro de volta pela rede para atualizar os pesos.

Descida do Gradiente: Conceito

- **O que é Descida do Gradiente?**
  - É um método de otimização utilizado para ajustar os pesos de uma rede neural de forma a minimizar a função de custo.
- **Intuição da Descida do Gradiente**
  - A descida do gradiente ajusta iterativamente os pesos na direção oposta ao gradiente da função de custo, reduzindo assim os erros da rede.
  - Cada passo na direção do gradiente negativo visa encontrar o ponto onde a função de custo atinge seu valor mínimo.
- **Objetivo Principal**
  - Minimizar a função de custo para melhorar a precisão das previsões da rede neural.

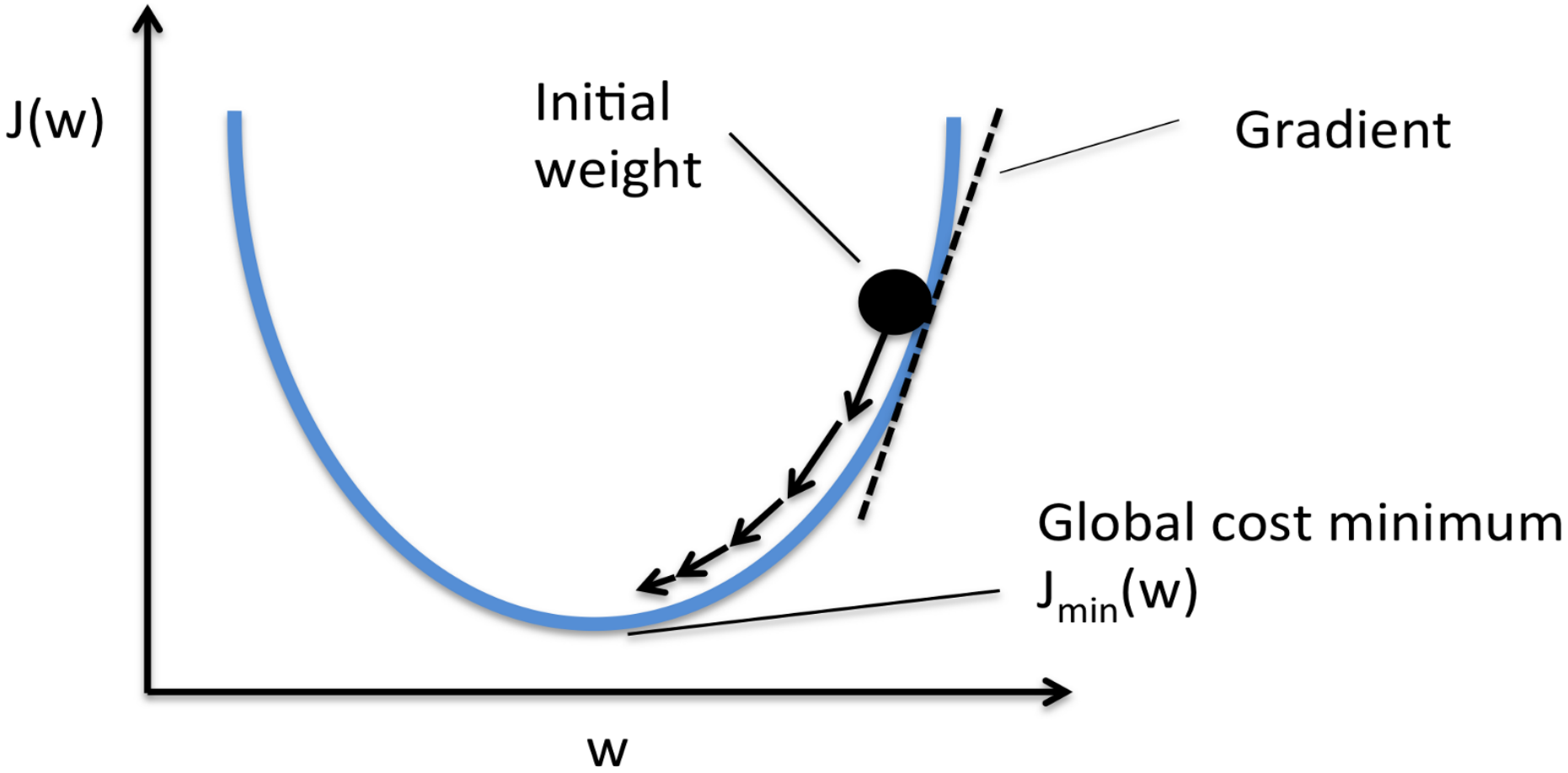
Descida do Gradiente: Derivadas Parciais

- **Derivadas Parciais**
  - Utilizadas para calcular o gradiente da função de custo em relação aos pesos.
  - Atualização dos pesos:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial J}{\partial w_{ij}}$$

  - Onde:
    - $w_{ij}^{(t)}$ é o peso na iteração t.
    - η é a taxa de aprendizado.
    - $\frac{\partial J}{\partial w_{ij}}$ é a derivada parcial da função de custo J em relação ao peso w_{ij}.

Ilustração da Descida do Gradiente



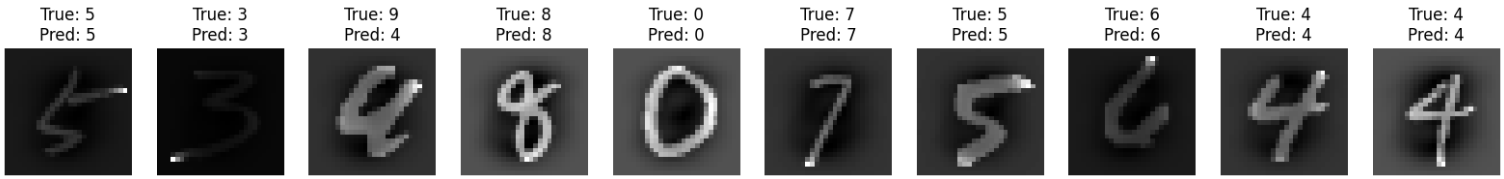
Implementação Prática

Implementação Prática

- **Estrutura da Rede Neural**
  - A rede neural é composta por camadas: entrada, ocultas e saída.
- **Processo de Treinamento**
  1. Inicialização dos pesos
  2. Forward propagation
  3. Cálculo da perda
  4. Backpropagation
  5. Atualização dos parâmetros
- **Código Principal**
  - Demonstração da função principal da rede neural.

Exemplo de Reconhecimento de Dígitos: Resultados

- **Detalhes do Código**
  - Veja o código completo no GitHub: digit-recognizer.ipynb
- **Exemplo de Previsões**



Conclusão

Conclusão

- Recapitulação dos principais pontos
- Aplicações futuras e possíveis melhorias
  - Existem otimizações que podem ser feitas em uma rede neural, como técnicas avançadas de inicialização de pesos e vieses.
  - Esta apresentação demonstrou a forma mais simples de uma rede neural, que já é capaz de realizar tarefas significativas.
  - Para mais detalhes sobre técnicas avançadas e implementações, consulte meu GitHub: github.com/felipeadeildo/neural-network

Obrigado!

Agradeço a todos pela atenção e interesse. É isso!