Relatório Técnico

Funções de Ativação em Redes Neurais:
Implementação e Análise Prática

Análise de Implementações em Python

27 de setembro de 2025

Conteúdo

1 Introdução

Este relatório apresenta uma análise detalhada das principais funções de ativação utilizadas em redes neurais, com foco em suas implementações práticas, propriedades matemáticas e aplicações em problemas de classificação. As funções estudadas incluem Sigmoid, Softmax, TanH, ReLU, LeakyReLU e ELU.

2 Funções de Ativação Implementadas

2.1 Função Sigmoid

2.1.1 Definição Matemática

A função sigmoid é uma das funções de ativação mais clássicas, definida por:

$$f(x) = \frac{1}{1 + e^{-x}}$$

2.1.2 Características Principais

- **Intervalo de saída**: $(0, 1)$
- **Não-linearidade**: Transforma entrada linear em saída não-linear
- **Interpretação probabilística**: Valores sempre positivos entre 0 e 1
- **Aplicação**: Classificação binária, camadas de saída

2.1.3 Implementação em Python

```
def funcao_sigmoide(x):
    """
    Funcao sigmoid
    Parametro: x - valor ou array de entrada
    Retorna: valor entre 0 e 1
    """
    return 1 / (1 + math.exp(-x))
```

2.1.4 Exemplo Prático - Detecção de Fraude

No documento, foi implementado um sistema de detecção de fraudes bancárias usando sigmoid com threshold de 0.5:

- Score > 0.5: Transação classificada como FRAUDE
- Score ≤ 0.5: Transação classificada como LEGÍTIMA

2.2 Função Softmax

2.2.1 Definição Matemática

A função softmax generaliza a sigmoid para múltiplas classes:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}$$

2.2.2 Características Principais

- **Intervalo de saída**: $(0, 1)$ para cada classe
- **Normalização**: Soma das probabilidades $= 1$
- **Aplicação**: Classificação multiclasse
- **Interpretação**: Distribuição de probabilidades

2.2.3 Implementação em Python

```
def softmax(x):
    """
    Funcao softmax para multiplas classes
    Parametro: x - array de scores (logits)
    Retorna: array de probabilidades normalizadas
    """
    exp_x = sy.exp(x)
    return exp_x / sy.Sum(exp_x)
```

2.2.4 Exemplo Prático - Classificação de Animais

Implementação para classificar imagens em três categorias:

- Cachorro: logit $= 2.0$

- Gato: logit = 1.0
- Pássaro: logit = 0.1

Após aplicar softmax: Cachorro (65.9%), Gato (24.2%), Pássaro (9.9%)

2.3 Função TanH (Tangente Hiperbólica)

2.3.1 Definição Matemática

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

2.3.2 Características Principais

- **Intervalo de saída**: $(-1, 1)$
- **Centrada em zero**: Facilita otimização
- **Saturação suave**: Evita mudanças bruscas
- **Aplicação**: Camadas ocultas, análise de sentimentos

2.3.3 Implementação em Python

```
def tanh(x):
    """
    Tangente hiperbolica
    Parametro: x - valor de entrada
    Retorna: valor entre -1 e 1
    """
    return (2 / (1 + sy.exp(-2*x))) - 1
```

2.3.4 Normalização para Probabilidade

Para converter saída tanh em probabilidade:

$$P = \frac{\tanh(x) + 1}{2}$$

2.4 Função ReLU (Rectified Linear Unit)

2.4.1 Definição Matemática

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

2.4.2 Características Principais

- **Simplicidade computacional**: Muito eficiente
- **Sem saturação positiva**: Gradientes não desaparecem
- **Esparsidade**: Neurônios inativos para $x < 0$
- **Problema**: "Dying ReLU- neurônios podem morrer permanentemente

2.4.3 Implementação em Python

```
def relu(x):
    """
    Funcao ReLU
    Parametro: x - valor ou array de entrada
    Retorna: max(0, x)
    """
    return sy.Max(0, x)
```

2.4.4 Análise de Ativação

No exemplo com CNN, observou-se que 55% dos neurônios permaneceram ativos após aplicar ReLU, demonstrando o efeito de esparsidade da função.

2.5 Função LeakyReLU

2.5.1 Definição Matemática

$$f(x) = \begin{cases} x & \text{se } x > 0 \\ \alpha x & \text{se } x \leq 0 \end{cases}$$

onde α é tipicamente 0.01 ou 0.3.

2.5.2 Características Principais

- **Evita "dying ReLU"**: Pequena inclinação negativa
- **Parâmetro α**: Controla inclinação negativa
- **Gradientes não-zero**: Mesmo para valores negativos

2.5.3 Implementação em Python

```
def leaky_relu(x, alpha=0.3):
    """
    Funcao LeakyReLU
    Parametros: x - entrada, alpha - inclinacao para x <
        0
    Retorna: x se x > 0, alpha*x caso contrario
    """
    if x > 0:
        return x
    else:
        return alpha * x
```

2.6 Função ELU (Exponential Linear Unit)

2.6.1 Definição Matemática

$$f(x) = \begin{cases} x & \text{se } x > 0 \\ \alpha(e^x - 1) & \text{se } x \leq 0 \end{cases}$$

2.6.2 Características Principais

- **Suavidade**: Exponencial na parte negativa
- **Saída média próxima de zero**: Facilita aprendizado
- **Sem "dying neurons"**: Gradientes sempre presentes

2.6.3 Implementação em Python

```
def elu(x, alpha=0.8):
    """
    Funcao ELU
    Parametros: x - entrada, alpha - parametro de escala
    Retorna: x se x > 0, alpha*(exp(x)-1) caso contrario
    """
    if x > 0:
        return x
    else:
        return alpha * (sy.exp(x) - 1)
```

3 Aplicações Práticas Implementadas

3.1 Sistema de Detecção de Fraudes

- **Função**: Sigmoid
- **Threshold**: 0.5
- **Resultado**: 2 de 5 transações classificadas como fraude

3.2 Análise de Sentimentos em E-commerce

- **Função**: Sigmoid para probabilidade de sentimento positivo
- **Entrada**: Scores de comentários
- **Saída**: Probabilidades de 10% a 97% para sentimento positivo

3.3 Classificação de Imagens com CNN

- **Função**: ReLU em camadas convolucionais
- **Resultado**: 55% de neurônios ativos após ReLU
- **Aplicação**: Detecção de características em 4 feature maps

3.4 Sistema de Controle de Motor

- **Função**: TanH para comandos suaves
- **Intervalo**: -1 (anti-horário) a +1 (horário)
- **Benefício**: Transições suaves evitam mudanças bruscas

4 Comparação entre Funções

Função	Intervalo	Centrada	Uso Principal
Sigmoid	$(0, 1)$	Não	Classificação binária
Softmax	$(0, 1)$	Não	Classificação multiclasse
TanH	$(-1, 1)$	Sim	Camadas ocultas
ReLU	$[0, \infty)$	Não	Camadas convolucionais
LeakyReLU	$(-\infty, \infty)$	Não	Evitar dying ReLU
ELU	$(-\alpha, \infty)$	Quase	Redes profundas

Tabela 1: Comparação das funções de ativação

5 Considerações de Implementação

5.1 Bibliotecas Utilizadas

- `numpy`: Operações vetorizadas
- `sympy`: Computação simbólica
- `matplotlib`: Visualização de gráficos

5.2 Boas Práticas Observadas

1. **Vetorização**: Uso de operações numpy para eficiência
2. **Modularização**: Funções bem definidas e reutilizáveis
3. **Validação**: Testes com múltiplos valores de entrada
4. **Visualização**: Gráficos para entender comportamento

6 Conclusão

As funções de ativação são componentes fundamentais em redes neurais, cada uma com características específicas adequadas para diferentes contextos:

- **Sigmoid/Softmax**: Ideais quando interpretação probabilística é necessária
- **TanH**: Útil quando valores centrados em zero são desejados

- **ReLU e variantes**: Escolha padrão para redes profundas devido à eficiência computacional

A escolha da função de ativação impacta diretamente no desempenho e convergência do modelo, sendo essencial considerar o problema específico e a arquitetura da rede.