Lista de Exercícios: Função ReLU em Redes Neurais Densas e Convolucionais

A função **ReLU (Rectified Linear Unit)** $f(x) = \max(0, x)$ é a função de ativação mais popular em deep learning devido à sua simplicidade computacional e eficiência no treinamento. Aqui estão 4 exercícios práticos cobrindo aplicações em redes densas e convolucionais.

Exercício 1: Classificação de Spam em Rede Neural Densa (MLP)

Cenário: Uma empresa desenvolveu um sistema de detecção de spam usando uma rede neural densa (multilayer perceptron). A camada oculta possui 5 neurônios especializados em detectar diferentes características dos emails. Após a combinação linear das entradas, cada neurônio produz um score que é processado pela função ReLU.

Outputs dos neurônios (antes da ativação):

- Neurônio 1: +2.3 (detectou palavras como "grátis", "promoção")
- Neurônio 2: -1.7 (detectou linguagem formal como "prezado", "cordialmente")
- Neurônio 3: +0.8 (detectou linguagem neutra)
- Neurônio 4: -0.2 (detectou estrutura de email normal)
- Neurônio 5: +4.1 (detectou muitos links suspeitos)

Tarefas:

a) Calcule $f(x) = \max(0, x)$ para cada neurônio

b) Determine quais neurônios ficaram ativos após ReLU

c) Se a camada seguinte usa a soma ponderada desses valores, calcule o valor total transmitido

Resolução Passo a Passo:

Passo 1: Aplicar ReLU a cada neurônio

Neurônio 1: $\text{ReLU}(2.3) = \max(0, 2.3) = 2.3$ → **ATIVO**

Neurônio 2: $\text{ReLU}(-1.7) = \max(0, -1.7) = 0.0$ → **INATIVO**

Neurônio 3: $\text{ReLU}(0.8) = \max(0, 0.8) = 0.8$ → **ATIVO**

Neurônio 4: $\text{ReLU}(-0.2) = \max(0, -0.2) = 0.0$ → **INATIVO**

Neurônio 5: $\text{ReLU}(4.1) = \max(0, 4.1) = 4.1$ → **ATIVO**

Passo 2: Análise da ativação

- **Neurônios ativos:** 3 de 5 (60% de ativação)
- **Neurônios mortos:** 2 de 5 (Neurônios 2 e 4)

Passo 3: Valor total transmitido
$$\text{Soma} = 2.3 + 0.0 + 0.8 + 0.0 + 4.1 = 7.2$$

Interpretação: O ReLU criou sparsidade na rede (40% dos neurônios inativos), focando apenas nas características mais relevantes para detecção de spam (palavras promocionais e links suspeitos).

Exercício 2: Detecção de Bordas em CNN (Primeira Camada Convolucional)

Cenário: Uma CNN para reconhecimento de imagens aplica um filtro de detecção de bordas verticais na primeira camada convolucional. A matriz de entrada representa uma região 5×5 de uma imagem com uma borda vertical clara.

Dados de entrada:

Matriz de entrada 5×5:

```
[0, 0, 1, 1, 1]
[0, 0, 1, 1, 1]
[0, 0, 1, 1, 1]
[0, 0, 1, 1, 1]
[0, 0, 1, 1, 1]
```

Kernel de detecção de borda vertical 3×3:

```
[-1, 0, 1]
[-1, 0, 1]
[-1, 0, 1]
```

Tarefas:

a) Calcule a convolução entre a matriz de entrada e o kernel (stride=1, sem padding)

b) Aplique ReLU aos resultados da convolução

c) Interprete o que a ReLU fez com os valores negativos

Resolução Passo a Passo:

Passo 1: Calcular a convolução 3×3

Para cada posição (i,j), calculamos:
$$\text{conv}(i,j) = \sum_{m=0}^{2} \sum_{n=0}^{2} \text{entrada}(i+m, j+n) \times \text{kernel}(m,n)$$

Posição (0,0):
$$\text{conv}(0,0) = (0 \times -1) + (0 \times 0) + (1 \times 1) + (0 \times -1) + (0 \times 0) + (1 \times 1) + (0 \times -1) +$$

Calculando todas as posições:

- Resultado da convolução:[1]

Passo 2: Aplicar ReLU
$ReLU() = $[1]

Passo 3: Interpretação

- **Valores positivos (3):** Detectaram a borda vertical (transição de 0 para 1)
- **Valores zero:** Regiões uniformes (sem bordas)
- **ReLU preservou** todos os valores pois não havia valores negativos neste caso

Feature Map resultante 3×3:

```
[3, 3, 0]
[3, 3, 0]
[3, 3, 0]
```

A ReLU manteve a detecção de bordas intacta, zerando apenas regiões sem características relevantes.

Exercício 3: Classificação de Dígitos com CNN (Feature Maps)

Cenário: Uma CNN para classificação de dígitos manuscritos (0-9) possui múltiplos feature maps após camadas convolucionais. Cada mapa especializa-se em detectar diferentes características: curvas, linhas horizontais, linhas verticais e cantos.

Feature maps antes da ativação:

- Mapa 1 (detecta curvas): [1.2, -0.5, 2.8, -1.1, 0.3]
- Mapa 2 (detecta linhas horizontais): [-0.8, 3.2, 0.1, -2.3, 1.7]
- Mapa 3 (detecta linhas verticais): [0.9, -1.4, -0.2, 2.5, -0.7]
- Mapa 4 (detecta cantos): [-3.1, 0.6, 1.8, -0.9, 2.2]

Tarefas:

a) Aplique ReLU a todos os feature maps

b) Conte quantos neurônios ficaram ativos em cada mapa

c) Analise qual característica foi mais detectada na imagem

Resolução Passo a Passo:

Passo 1: Aplicar ReLU a cada feature map

Mapa 1 (Curvas):
$$\text{ReLU}([1.2, -0.5, 2.8, -1.1, 0.3]) = [1.2, 0.0, 2.8, 0.0, 0.3]$$

- **Neurônios ativos:** 3/5

Mapa 2 (Linhas Horizontais):
$\text{ReLU}([-0.8, 3.2, 0.1, -2.3, 1.7]) = [0.0, 3.2, 0.1, 0.0, 1.7]$

- **Neurônios ativos:** 3/5

Mapa 3 (Linhas Verticais):
$\text{ReLU}([0.9, -1.4, -0.2, 2.5, -0.7]) = [0.9, 0.0, 0.0, 2.5, 0.0]$

- **Neurônios ativos:** 2/5

Mapa 4 (Cantos):
$\text{ReLU}([-3.1, 0.6, 1.8, -0.9, 2.2]) = [0.0, 0.6, 1.8, 0.0, 2.2]$

- **Neurônios ativos:** 3/5

Passo 2: Análise de ativação

Feature Map	Neurônios Ativos	Intensidade Total	Característica
Mapa 1 (Curvas)	3/5 (60%)	4.3	Bem detectada
Mapa 2 (Linhas H.)	3/5 (60%)	5.0	**Mais detectada**
Mapa 3 (Linhas V.)	2/5 (40%)	3.4	Moderadamente detectada
Mapa 4 (Cantos)	3/5 (60%)	4.6	Bem detectada

Passo 3: Interpretação

- **Característica mais presente:** Linhas horizontais (maior intensidade total)
- **ReLU eliminou ruído** (valores negativos) mantendo apenas características positivamente detectadas
- **Sparsidade:** Aproximadamente 50% dos neurônios inativos, focando processamento nas características relevantes

Exercício 4: Análise do Problema "Dying ReLU"

Cenário: Durante o treinamento de uma rede neural, alguns neurônios podem "morrer" quando seus pesos se ajustam de forma que sempre produzem valores negativos. Este exercício analisa a evolução de 5 neurônios ao longo de 40 épocas de treinamento.

Evolução dos outputs (antes da ReLU):

Época	Neurônio 1	Neurônio 2	Neurônio 3	Neurônio 4	Neurônio 5
1	+1.5	+0.8	-0.2	+2.1	-1.3
10	+0.3	-0.8	-1.1	+0.7	-2.1
20	-0.1	-1.5	-2.0	-0.5	-3.2
30	-1.2	-2.1	-2.8	-1.8	-4.1
40	-2.5	-3.2	-3.7	-2.9	-5.0

Tarefas:

a) Calcule as saídas ReLU para cada época

b) Determine quando ocorre o "dying ReLU"

c) Explique por que isso é problemático para o aprendizado

Resolução Passo a Passo:

Passo 1: Aplicar ReLU e contar neurônios ativos

Época	Outputs Pós-ReLU	Neurônios Ativos	Status
1	[1.5, 0.8, 0.0, 2.1, 0.0]	3/5 (60%)	**Saudável**
10	[0.3, 0.0, 0.0, 0.7, 0.0]	2/5 (40%)	**Problema**
20	[0.0, 0.0, 0.0, 0.0, 0.0]	0/5 (0%)	**DYING ReLU!**
30	[0.0, 0.0, 0.0, 0.0, 0.0]	0/5 (0%)	**DYING ReLU!**
40	[0.0, 0.0, 0.0, 0.0, 0.0]	0/5 (0%)	**DYING ReLU!**

Passo 2: Análise da evolução

Época 1-10: Degradação gradual

- Neurônios ainda contribuem para o aprendizado
- Gradientes ainda fluem durante backpropagation

Época 20 em diante: Dying ReLU completo

- **Todos os neurônios mortos:** Output sempre zero
- **Gradiente zero:** $\frac{\partial \text{ReLU}}{\partial x} = 0$ quando $x < 0$
- **Pesos congelados:** Sem atualizações nos pesos

Passo 3: Por que é problemático?

Problemas do Dying ReLU:

1. **Perda de Capacidade:** Neurônios mortos não contribuem para a representação
2. **Gradiente Zero:** $\nabla_w = 0$ impede atualizações de pesos
3. **Irreversibilidade:** Neurônios raramente "ressuscitam"
4. **Redução da Expressividade:** Rede efetivamente menor

Soluções Possíveis:

- **Leaky ReLU:** $f(x) = \max(\alpha x, x)$ onde $\alpha = 0.01$
- **Parametric ReLU:** α aprendido durante treinamento
- **Inicialização adequada:** He initialization para pesos
- **Learning rate menor:** Evita atualizações muito drásticas

Propriedades Fundamentais da ReLU

Definição Matemática:

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

Derivada:

$$f'(x) = \begin{cases} 1 & \text{se } x > 0 \\ 0 & \text{se } x \leq 0 \end{cases}$$

Valores de Referência:

Input (x)	ReLU f(x)	Derivada f'(x)	Status
-3	0	0	Inativo
-1	0	0	Inativo
0	0	0*	Limiar
1	1	1	Ativo
3	3	1	Ativo
5	5	1	Ativo

*Na prática, define-se f'(0) = 0 ou f'(0) = 1 dependendo da implementação.

Vantagens da ReLU:

1. **Eficiência Computacional:** Operação simples ($\max(0, x)$)
2. **Evita Vanishing Gradient:** Gradiente constante (1) para valores positivos
3. **Sparsidade Natural:** Aproximadamente 50% dos neurônios inativos
4. **Convergência Rápida:** Permite treinamento eficiente de redes profundas
5. **Interpretabilidade:** Ativações representam "intensidade" de características

Quando Usar ReLU:

- **Redes Densas:** Camadas ocultas de MLPs
- **CNNs:** Após camadas convolucionais e fully-connected
- **Problemas de Classificação:** Especialmente com muitas classes
- **Redes Profundas:** Onde vanishing gradient é preocupação

Limitações:

1. **Dying ReLU Problem:** Neurônios podem "morrer" permanentemente
2. **Não Zero-Centrada:** Saída sempre não-negativa
3. **Unbounded Output:** Pode levar a exploding gradients
4. **Não Diferenciável em x=0:** Problema teórico, não prático

A função ReLU revolucionou o deep learning por sua simplicidade e eficácia, sendo a escolha padrão para a maioria das arquiteturas modernas de redes neurais!

⁂

1. https://arxiv.org/abs/2010.13572
2. https://infoslack.pro/ml-book/contents/cnn-tensorflow.html
3. https://www.baeldung.com/cs/ml-relu-dropout-layers
4. https://www.kaggle.com/code/aisuko/training-a-dense-neural-network
5. https://ele.ufes.br/sites/engenhariaeletrica.ufes.br/files/field/anexo/kaio_g_utsch.pdf
6. https://www.sciencedirect.com/science/article/pii/S2666827023000555
7. https://www.cloudskillsboost.google/course_templates/18/video/381962?locale=it
8. https://www.deeplearningbook.com.br/reconhecimento-de-imagens-com-redes-neurais-convolucionais-em-python-parte-4/
9. https://www.youtube.com/watch?v=68BZ5f7P94E
10. https://www.youtube.com/watch?v=0IvHURoyhtc
11. https://www.inf.ufg.br/~anderson/deeplearning/20181/Aula - Redes Neurais Convolucionais Parte I.pdf
12. https://developers.google.com/machine-learning/crash-course/neural-networks/interactive-exercises?hl=pt-br
13. https://www.passeidireto.com/arquivo/120937355/deep-learning
14. https://www.codecademy.com/article/rectified-linear-unit-relu-function-in-deep-learning
15. https://abjur.github.io/r4jurimetrics/redes-neurais-convolucionais.html
16. https://encord.com/blog/activation-functions-neural-networks/
17. https://d2l.ai/chapter_multilayer-perceptrons/mlp.html
18. https://www.ime.unicamp.br/~jbflorindo/Teaching/2018/MT530/T10.pdf
19. https://www.geeksforgeeks.org/deep-learning/relu-activation-function-in-deep-learning/
20. https://www.kaggle.com/code/ryanholbrook/exercise-deep-neural-networks