Diferenciando o Não Diferenciável: Investigando problemas na diferenciação da ReLU no treinamento de modelos de Aprendizado Profundo

Roberto Sprengel Minozzo Tomchak
Departamento de Informática
Universidade Federal do Paraná
Curitiba, Paraná, Brasil
robertotomchak@ufpr.br

Lucas Garcia Pedroso
Departamento de Matemática
Universidade Federal do Paraná
Curitiba, Paraná, Brasil
lucaspedroso@ufpr.br

Paulo Ricardo Lisboa de Almeida
Departamento de Informática
Universidade Federal do Paraná
Curitiba, Paraná, Brasil
paulorla@ufpr.br

ABSTRACT

Activation functions are necessary for deep learning models to be able to represent non-linear relationships. Their derivative is required during training, however, many non differentiable activation functions are commonly used in neural networks, such as the *Rectified Linear Unit* (ReLU) and its variants. This paper discusses the impact of non differentiability on activation functions during the training phase, and how these functions compare to differentiable alternatives. To analyse this problem, we trained neural networks in an image classification problem using various activation functions. We showed that non-differentiable points occur rarely during training, especially in deep models, and have little to no negative impact in a model's performance.

KEYWORDS

Aprendizado Profundo, ReLU, Funções de ativação, Retropropagação

1 INTRODUÇÃO

Redes neurais têm como objetivo calcular determinados valores alvo, baseados numa entrada. Esses valores alvo podem ser probabilidades de pertencer a uma classe [1], valores brutos [2], regiões numa imagem [3] ou outros, a depender do problema de interesse. Para realizar essa tarefa, os neurônios de uma rede com múltiplas camadas calculam seus valores $\hat{y}$ baseados na Equação 1, onde $\mathbf{w}$ é o vetor de pesos do neurônio, b é seu viés, f é uma função de ativação e $\mathbf{x}$ são os valores dos neurônios da camada anterior. Para treinar essas redes, é comum o uso do algoritmo de retropropagação [4] com descida de gradiente usando fator de aprendizagem η, onde uma função de perda E determina o erro no cálculo dos neurônios, e seus pesos e vieses são atualizados conforme as Equações 2 e 3. Ou seja, os valores de $\mathbf{w}$ e b que serão utilizados na iteração $\tau + 1$ são atualizados em função de seus valores na iteração atual τ, subtraindo-se o gradiente do erro [5].

$$\hat{y} = f(\mathbf{w}\mathbf{x} + b) \tag{1}$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{2}$$

$$b^{(\tau+1)} = b^{(\tau)} - \eta \nabla E(b^{(\tau)}) \tag{3}$$

Para obter ∇E, é necessário conhecer a derivada da função de ativação f. Portanto, é imprescindível que f seja uma função derivável. Entretanto, o uso de funções que não são diferenciáveis em alguns pontos do seu domínio é prática bastante comum. Essa escolha se dá por essas funções tenderem a ter um custo computacional menor, além de permitirem o treinamento de redes profundas, com centenas de camadas [6].

Por exemplo, considere a função ReLU, introduzida no âmbito dos perceptrons no artigo de Fukushima [7], de uso popularizado em Redes Neurais Profundas – *Deep Neural Network*s (DNNs) [8]. Esta função é o foco deste trabalho e tem comportamento exemplificado na Figura 1. Note que no ponto $x = 0$ a inclinação da função à esquerda é diferente da sua inclinação à direita, fazendo com que sua derivada nesse ponto não esteja bem definida.

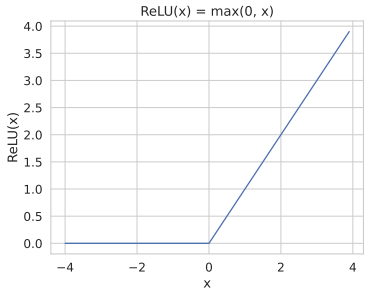

Figura 1: Exemplo da Função ReLU.

Dessa forma, surge-se a questão de como esses pontos de não diferenciabilidade de funções de ativação impactam no treinamento das redes neurais, e se a escolha de funções diferenciáveis em todo seu domínio poderia trazer benefícios. Este trabalho busca compreender esse problema, ao treinar e afinar redes profundas para processamento de imagens usando funções não diferenciáveis, como a ReLU e suas equivalentes diferenciáveis, como a *Gaussian Error Linear Units* (GELU). O objetivo deste trabalho é responder as seguintes perguntas de pesquisa:

(1) Com que frequência pontos de não diferenciabilidade são atingidos durante o treinamento dessas redes?
(2) Qual o custo computacional extra envolvido ao se usar funções de ativação similares, mas diferenciáveis, e quais os benefícios quanto à convergência?
(3) Como o tamanho da rede pode afetar esses resultados?

Nosso trabalho mostra que esses pontos de não diferenciabilidade surgem raramente durante o treinamento, especialmente em modelos com muitas camadas, e não impactam a performance do modelo. Para provar essa hipótese, organizamos o restante deste trabalho da seguinte forma. Na Seção 2, são apresentadas as funções de ativação deste artigo. Na Seção 3, é discutido o artigo de Bertoin et al. [9], o qual mostra a importância da definição do valor da derivada da ReLU em seu ponto de não diferenciabilidade, além do artigo de Hendrycks e Gimpel [10], que discute sobre o uso da ReLU e sua versão diferenciável, a GELU. Na Seção 4, é descrito o protocolo experimental usado para responder as perguntas de pesquisa, e na Seção 5 são apresentados os resultados dos experimentos. Finalmente, na Seção 6 são apresentadas as conclusões obtidas nos experimentos. [1]

2 FUNÇÕES DE ATIVAÇÃO

Historicamente, a função Sigmoid ($f(x) = 1/(1+exp(-x))$) foi uma das primeiras funções de ativação usadas em redes neurais [11], devido à sua similaridade com a forma como neurônios transmitem informação no cérebro humano. Entretanto, diversos problemas surgiram no treinamento de redes usando essa função. O principal deles sendo o *vanishing gradients* [12], onde a atualização dos pesos dos gradientes tende a diminuir ao longo do processo de retropropagação, fazendo com que as primeiras camadas da rede sejam pouco atualizadas durante o treinamento. Isso fez com que o treinamento de redes profundas fosse inviável.

Uma técnica que surgiu para lidar com o problema do *vanishing gradients* foi a introdução da função ReLU como função de ativação [13]. A Equação 4 descreve a ReLU. Como essa função realiza seus cálculos simplesmente se baseando no sinal do seu valor de entrada, sua implementação é computacionalmente eficiente [14]. A derivada da ReLU é descrita pela Equação 5. Note que a derivada não é definida no ponto $x = 0$, portanto, este é o ponto de não diferenciabilidade da função. Na prática, diversas APIs costumam definir que $\text{ReLU}'(0) = 0$ [9].

$$\text{ReLU}(x) = \begin{cases} x, & \text{se } x \geq 0, \\ 0, & \text{se } x < 0 \end{cases} \tag{4}$$

$$\text{ReLU}'(x) = \begin{cases} 1, & \text{se } x > 0, \\ 0, & \text{se } x < 0 \end{cases} \tag{5}$$

Devido ao problema da não diferenciabilidade da ReLU, surgiu a ideia de usar funções que possuem curva similar à da ReLU, mas que são totalmente diferenciáveis. Uma dessas funções é a GELU, introduzida em [10]. A Equação 6 descreve a GELU, enquanto sua derivada é descrita pela Equação 7. Comparando com a ReLU, a GELU é mais complexa de ser calculada, porém tem a vantagem de ser diferenciável em todo o seu domínio [15]. As duas funções de ativação possuem valores similares, como pode ser visto na figura 2.

$$\text{GELU}(x) = x \cdot \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt \tag{6}$$

[1] Os códigos utilizados para executar os experimentos, bem como as análises realizadas sobre eles, então disponíveis em https://github.com/robertotomchak/ReLUDiffStudy.

$$\text{GELU}'(x) = x \cdot \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} + \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt \tag{7}$$

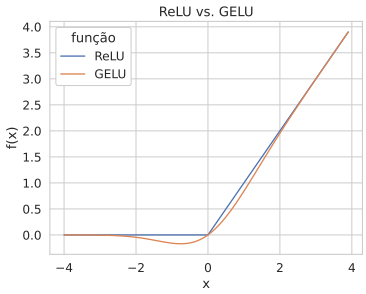

Figura 2: Comparação das funções ReLU e GELU.

Outras funções de ativação podem ser usadas para resolver outros tipos de problemas. Neste artigo, também são consideradas outras duas funções de ativação não diferenciáveis: a Hardsigmoid [16] e a Hardswish [17], descritas nas Equações 8 e 9, respectivamente. Observe que elas possuem dois pontos de não diferenciabilidade ($x = -3$ e $x = 3$). A Hardsigmoid é uma forma de aproximar a função Sigmoid com uma equação mais computacionalmente eficiente de calcular. Similarmente, a Hardswish tem como objetivo aproximar a função Swish ($f(x) = x/(1+exp(-x))$). As Figuras 3 e 4 mostram as funções Hardsigmoid e Hardswish, respectivamente, bem como suas versões diferenciáveis.

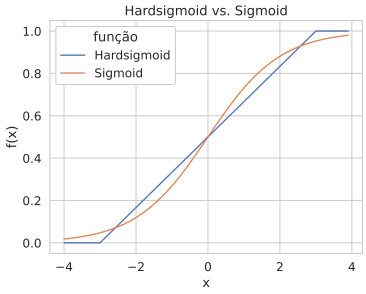

Figura 3: Comparação das funções Hardsigmoid e Sigmoid.

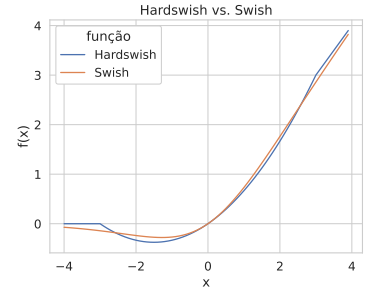

Figura 4: Comparação das funções Hardswish e Swish.

$$\text{Hardsigmoid}(x) = \begin{cases} 0, & \text{se } x \leq -3, \\ 1, & \text{se } x \geq +3, \\ \frac{x}{6} + \frac{1}{2}, & \text{caso contrário} \end{cases} \quad (8)$$

$$\text{Hardswish}(x) = \begin{cases} 0, & \text{se } x \leq -3, \\ x, & \text{se } x \geq +3, \\ \frac{x \cdot (x+3)}{6}, & \text{caso contrário} \end{cases} \quad (9)$$

3 TRABALHOS RELACIONADOS

Em Bertoin et al. [9], os autores discutem sobre a escolha do valor de ReLU′(0), e como essa escolha impacta no treinamento da rede. Considere que $s = \text{ReLU}'(0)$, ou seja, s é o valor definido para a derivada da ReLU em seu ponto de não diferenciabilidade. Em teoria, o valor de s pode ser qualquer valor no intervalo $[0, 1]$, devido à definição de subgradiente [18]. Além disso, como a ReLU é definida para todos os números reais, a probabilidade de ser necessário computar ReLU'(0) durante o treinamento deveria ser negligenciável.

Entretanto, como computadores realizam cálculos de valores reais usando representação em ponto flutuante, esses valores já não são contínuos e são representados num intervalo discreto, levando a imprecisões de cálculo. Dessa forma, a chance de ser necessário utilizar o valor s se torna maior. Os autores mostram que a escolha entre usar $s = 0$ ou $s = 1$ pode levar a divergências nos pesos da rede após algumas épocas. Dentre os possíveis valores de s, foi mostrado que usar $s = 0$ leva a uma melhor acurácia em teste, devido a valores $s \neq 0$ levarem a um comportamento caótico na função de perda da rede durante o treinamento.

Esse estudo clareou como a escolha de ReLU′(0) impacta no treinamento de redes neurais. Entretanto, o estudo não mostra como essa função não diferenciável se compara com alternativas diferenciáveis, como a GELU. Além disso, o artigo, na Seção 4.5, realiza um experimento onde a proporção de chamadas de ReLU′(0) em relação ao total de chamadas de ReLU′(x) parece ser constante, independentemente do tamanho da rede. Nosso trabalho planeja compreender melhor essa relação.

Um estudo que discute como a ReLU e a GELU se comparam é o de Hendrycks e Gimpel [10]. Este é o trabalho que introduziu a GELU e comparou as duas funções de ativação em diversos problemas de redes neurais. Os autores mostram que a GELU tende a atingir acurácias maiores que a ReLU, embora essa diferença pareça ser relativamente pequena. Apesar do artigo jogar luz sobre como essas funções de ativação impactam na acurácia da rede, falta compreender como essas funções se comparam em relação a outras métricas, como o tempo de treinamento e de previsão.

Assim, nosso trabalho foca em complementar os resultados desses dois estudos, abordando o impacto dos pontos de não diferenciabilidade no treinamento dessas redes.

4 PROTOCOLO EXPERIMENTAL

Para responder as perguntas de pesquisa, são consideradas redes treinadas para a tarefa de classificação de imagens, por ser um problema clássico de aprendizado profundo. Para comparar os tipos de funções de ativação, as redes têm duas versões: modelo diferenciável, que só usa funções de ativação diferenciáveis em seus domínios, e modelo não diferenciável, que só usa funções de ativação que possuem pelo menos um ponto de não diferenciabilidade. Para avaliar como o tamanho das redes impacta na frequência de pontos de não diferenciabilidade durante o treinamento, as redes são também modificadas para versões com número de parâmetros e de camadas diferentes.

Para evitar divergências de resultados, os experimentos são executados 5 vezes, com fixação da semente do gerador de números pseudoaleatórios para experimentos que envolvem a mesma rede. Para a análise dos resultados, são consideradas as médias das informações coletadas por execução.

Os testes são executados em ambiente Python, com a biblioteca Pytorch na versão 2.4.1 e o CUDA versão 12.0. A máquina utilizada possui 2 CPUs modelo Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz, cada uma com 12 *cores* e frequência base de 2.20 GHz, GPU NVIDIA RTX A5000 e memória DRAM de 128 GB, e uso do sistema operacional Linux Ubuntu 23.10.

Tabela 1: Divisão das imagens do FashionMNIST.

Dataset	Quantidade	% (do total)
Treino	42.000	60%
Validação	18.000	25,71%
Teste	10.000	14,29%
Total	70.000	100%

4.1 Dataset

Para o treinamento e teste das redes, é utilizado o dataset FashionMNIST [19], considerado um benchmark para algoritmos de aprendizado de máquina. O dataset é similar ao MNIST original [20], contendo imagens 28x28 em escala cinza, porém com a diferença que as imagens representam uma de 10 possíveis peças de roupas (como vestidos e calçados). A Tabela 1 mostra a divisão das imagens entre treino, validação e teste.

4.2 Redes Neurais

A principal rede desse estudo, nomeada de rede de Hochuli, possui arquitetura definida em [21] e foi desenvolvida para a classificação de carros em imagens de câmeras de estacionamento [22]. As imagens usadas nesse artigo para treinar essa rede possuem escala pequena (32×32), sendo uma arquitetura razoável para lidar com o FashionMNIST. Essa rede é pequena, com cerca de 70 mil parâmetros, divididos em 3 camadas de convolução mais 2 camadas no *fully connected*. A rede de Hochuli usa originalmente a ReLU como função de ativação, modificada em nossos experimentos para a GELU na versão diferenciável da rede. A arquitetura original da rede pode ser vista na Figura 5.

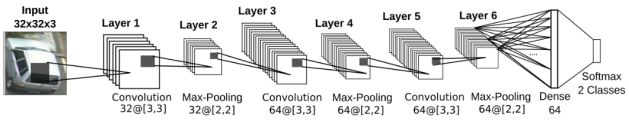

Figura 5: Arquitetura Original da Rede de Hochuli.

Além disso, foram treinadas duas variações dessa rede, para analisar como a frequência de pontos de não diferenciabilidade está relacionada com o tamanho da rede. A primeira, chamada de Hochuli Dobrada, possui a mesma arquitetura que a Hochuli original, mas com o número de filtros nas camadas de convolução dobrados. Por exemplo, se uma camada tinha 32 filtros de saída na rede original, terá 64 na versão dobrada. Dessa forma, temos uma rede com o mesmo número de camadas, mas agora com mais parâmetros, subindo para cerca de 257 mil parâmetros. A outra, chamada de Hochuli Profunda, têm as camadas de *Max Pooling* da rede de Hochuli modificadas para camadas de convolução com filtros 2x2 e *stride* de 2. Dessa forma, as dimensões dos filtros entre camadas se mantêm iguais, mas como as camadas de *Max Pooling* não possuem parâmetros treináveis, isso tem o efeito de dobrar o número de camadas da rede na sua parte convolucional, sem alterar o *fully connected* no final. Assim, a rede foi de 3 camadas de convolução para 6 camadas de convolução, com o número de parâmetros subindo para cerca de 110 mil.

Para avaliar como funções de ativação não diferenciáveis afetam um modelo estado da arte com maior número de parâmetros e camadas, os experimentos foram repetidos para a rede MobileNetV3 [23], modelo desenvolvido para ser usado em aplicações de dispositivos móveis. Para este estudo, é usada a versão *small* da rede, que possui cerca de 1,5 milhão de parâmetros, por ser considerado um tamanho adequado para ser treinada no hardware utilizado. Além disso, a rede faz o uso de três funções de ativação: a ReLU, a Hardsigmoid e a Hardswish. Para a versão diferenciável, essas funções de ativação foram trocadas, respectivamente, por GELU, Sigmoid e Swish. Como a rede possui um número de parâmetros muito maior que o número de imagens dos dados de treino, é usada a rede pré-treinada com todas as camadas descongeladas. A arquitetura original da rede pode ser vista na Figura 6.

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Figura 6: Arquitetura Original da MobileNetV3 *small*.

Todas as redes tiveram a última camada modificada para ter 10 neurônios de saída, pois é o número de classes do FashionMNIST, e foram treinadas por 50 épocas, mais uma iteração sobre o *dataset* de teste no final.

4.3 Função de Perda e Otimizador

Para treinar os modelos, é usada como função de perda a *Cross Entropy*, comumente utilizada para tarefas de classificação [5, 10, 21]. Já o otimizador usado é o *Stochastic Gradient Descent* (SGD) com fator de aprendizagem de 0, 001, valor padrão da biblioteca do Pytorch. O objetivo dessa escolha é reduzir o número de hiperparâmetros que precisam ser ajustados, para diminuir o número de variáveis que poderiam afetar os resultados dos experimentos.

Também foi usado um momento de 0,9 no otimizador. As Equações 10 e 11 mostram como o algoritmo de SGD atualiza os pesos e vieses, onde μ é o momento e $\Delta\mathbf{w}^{(\tau-1)}$ é a atualização do vetor $\mathbf{w}$ na etapa anterior, com raciocínio análogo para $\Delta b^{(\tau-1)}$ [5]. Como os pesos da iteração seguinte dependem da atualização dos pesos das iterações anteriores, problemas provenientes do cálculo de derivadas em pontos de não diferenciabilidade numa iteração são carregados nas iterações seguintes. Assim, a escolha de um momento alto força um caso mais complicado para o uso das funções de ativação não diferenciáves, e portanto é possível avaliar melhor o impacto dos pontos de não diferenciabilidade no treinamento do modelo, além de ser um valor clássico na literatura [5].

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{(\tau)}) + \mu\Delta\mathbf{w}^{(\tau-1)} \tag{10}$$

$$b^{(\tau+1)} = b^{(\tau)} - \eta\nabla E(b^{(\tau)}) + \mu\Delta b^{(\tau-1)} \tag{11}$$

4.4 Informações Analisadas

A cada época e iteração de teste, são coletadas as seguintes informações:

(1) Contagem de chamadas ao gradiente da função de ativação. No caso da ReLU, seria $\text{ReLU}'(x)$, para qualquer x;
(2) Contagem de chamadas ao gradiente da função de ativação num ponto dentro de uma certa tolerância de distância de um ponto de não diferenciabilidade. No caso da ReLU, seria $\text{ReLU}'(x)$, para todo x tal que $|x - 0| \leq \epsilon$, onde ϵ é uma tolerância definida;
(3) Tempo total da época ou iteração de teste (em segundos);
(4) Perda de treino e validação;
(5) Perda e acurácia de teste.

As duas primeiras informações são usadas para determinar a proporção das chamadas de cálculo de gradiente que foram em pontos de não diferenciabilidade. Neste artigo, definimos que a proporção de não diferenciabilidade é a divisão entre o número de chamadas ao gradiente da função de ativação num ponto dentro de uma certa tolerância de distância de um ponto de não diferenciabilidade pelo número total de chamadas ao gradiente. Ou seja, é a divisão da segunda informação coletada pela primeira. A análise do tempo ajuda a entender o quão rápido esses modelos convergem, bem como a velocidade de suas previsões. A perda é coletada para avaliar a qualidade das previsões dos modelos e compreender como as funções de ativação afetam a convergência das redes.

A contagem de chamadas ao gradiente e seus pontos de não diferenciabilidade é coletada usando funções customizadas, desenvolvidas com a API do Pytorch. Essas funções são matematicamente equivalentes às funções de ativação originais, mas com o adicional que, sempre que o cálculo do seu gradiente é realizado, um contador

global é incrementado, e se esse gradiente é calculado sobre um ponto dentro da tolerância de um ponto de não diferenciabilidade, um outro contador global de gradientes não bem definidos é incrementado também. Dessa forma, essas funções nos permitem descobrir quantas vezes o gradiente foi calculado e quantos desses foram em pontos de não diferenciabilidade. Como essas funções customizadas são mais lentas que as funções *built-in* do Pytorch, o tempo de execução do modelo não diferenciável é avaliado usando as funções *built-in* do Pytorch.

Como números decimais são representados com imprecisão em ponto flutuante, foi considerado como ponto de não diferenciabilidade qualquer valor a uma certa tolerância de distância de um ponto de não diferenciabilidade real da função. Ou seja, qualquer valor x tal que $|x - p| \leq \epsilon$ é considerado um ponto de não diferenciabilidade em nossos experimentos, onde p é um ponto de não diferenciabilidade real da função. No caso da ReLU, $p = 0$. A tolerância usada nesse experimento é o épsilon da máquina, definido como a diferença entre o 1 e o menor valor representável que seja maior que 1 [24]. Para os experimentos, os cálculos são realizados com valores de 32 bits de precisão no padrão IEEE 754, cujo épsilon é igual à 2^{-23}.

5 EXPERIMENTOS E RESULTADOS

Inicialmente, são apresentados os resultados sobre a rede de Hochuli nas Seções 5.1 e 5.2. Em seguida, na Seção 5.3 são comparadas as diferentes versões da rede de Hochuli, para compreender o impacto do tamanho da rede sobre os resultados. Por fim, as análises são repetidas para a rede MobileNetV3 na Seção 5.4.

5.1 Frequência do Ponto de não diferenciabilidade

A Figura 7 mostra a proporção de não diferenciabilidade por época no treinamento da rede de Hochuli, usando a ReLU como função de ativação. Note que a escala do eixo y está em 10^{-5}, portanto, essa proporção é muito baixa. É possível perceber também que, embora há um padrão de queda da proporção durante o treino, a curva ainda é bem caótica. No total, ao longo do treinamento inteiro, o gradiente da ReLU foi calculado 52.819.200.000 vezes, sendo que 403.850 dessas chamadas foram dentro da tolerância em relação ao ponto $x = 0$. Assim, uma a cada 130.789 chamadas de gradiente da ReLU foi num ponto de não diferenciabilidade.

5.2 Modelo Não Diferenciável vs. Diferenciável

A Figura 8 compara o valor da as curvas de perda (*loss*) nos dados de validação para a rede de Hochuli usando a ReLU e a GELU. É possível perceber que os modelos têm desempenho muito similar. Embora a diferença seja pequena, o modelo diferenciável atingiu a menor perda nos dados de validação na época 31, enquanto o não diferenciável precisou de mais 2 épocas, atingindo o mínimo na época 33.

Os parâmetros da época com menor perda na validação foram usados para avaliar o modelo nos dados de teste. A Tabela 2 mostra a comparação dos modelos. O modelo com funções diferenciáveis converge algumas épocas antes, o que é esperado, considerando que o gradiente dessas funções sempre está definido. Assim, sempre é possível usar o gradiente para ir em direção ao ponto de mínimo, ao

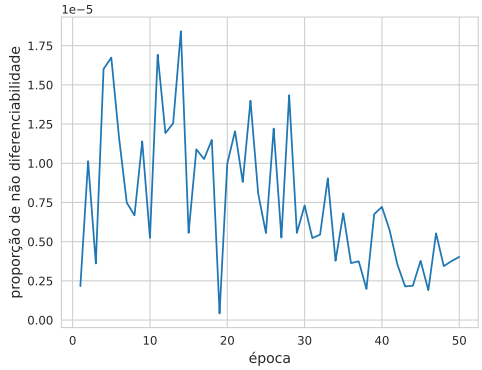

Figura 7: Proporção de não diferenciabilidade da ReLU por época.

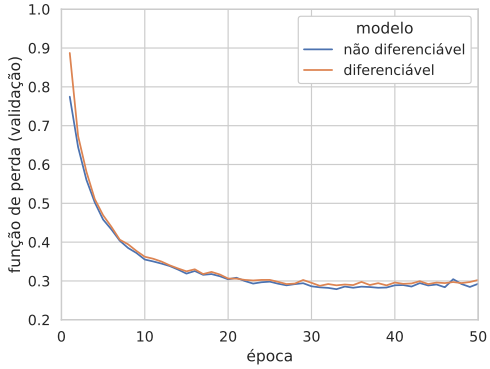

Figura 8: Função de perda na validação da rede de Hochuli por época.

contrário do modelo não diferenciável, onde pontos de não diferenciabilidade evitam o cálculo correto do gradiente para minimizar a função. O tempo médio por época dos modelos é muito similar. Embora seja esperado que funções de ativação diferenciáveis tenham cálculo mais custoso em relação às funções não diferenciáveis, por terem fórmulas mais computacionalmente complexas, uma hipótese para os modelos terem obtido tempos similares é que diversas otimizações feitas na GPU deixam os tempos de ambos os modelos similares, além de que a ReLU pode ter dependências de desvio, devido a sua natureza condicional. Ainda, é provável que os outros cálculos da rede, como a multiplicação de matrizes, sejam muito mais pesados do que os cálculos das funções de ativação e seus gradientes, mascarando essa diferença. Entretanto, são necessários trabalhos futuros para confirmar tais hipóteses. Quanto ao desempenho nos dados de teste, os dois modelos apresentam resultados similares: a acurácia atingida por ambos é praticamente a mesma e o modelo diferenciável é apenas 4% mais lento para determinar a classe das imagens nos dados de teste.

Tabela 2: Métricas dos modelos com rede de Hochuli na melhor época.

Métrica	Não Dif.	Dif.
Melhor Época	33	31 (-6,06%)
Acurácia (teste)	89,93%	89,86% (-0.07%)
Tempo Médio por Época (s)	13,41	13,31 (-0,72%)
Tempo Médio de Previsão (s)	$1,71 \cdot 10^{-4}$	$1,78 \cdot 10^{-4}$ (4,18%)

Tabela 3: Comparação das chamadas de gradiente da ReLU nas versões da rede de Hochuli.

	Original	Dobrada	Profunda
Parâmetros	73.418	256.714	110.442
Camadas	5	5	8
Gradientes Não Dif.	403.850	878.824	23.355
Gradientes Totais	$5,28 \cdot 10^{10}$	$1,06 \cdot 10^{11}$	$5,28 \cdot 10^{10}$
Proporção de Não Dif.	$7,65 \cdot 10^{-6}$	$8,33 \cdot 10^{-6}$	$4,42 \cdot 10^{-7}$

5.3 O Impacto da Arquitetura da Rede nos Resultados

A Tabela 3 mostra o número de chamadas do gradiente da ReLU e sua relação com o número de camadas e parâmetros das três versões da rede de Hochuli. A linha "Gradientes Não Dif." mostra o número de chamadas ao gradiente das função de ativação em pontos dentro da tolerância de um ponto de não diferenciabilidade ao longo de todo o treino. Similarmente, a linha "Gradientes Totais" mostra o número total de chamadas do gradiente da função de ativação, em qualquer ponto. Note que da rede original para sua versão dobrada a proporção de não diferenciabilidade muda pouco, mesmo com o número de parâmetros quase quadruplicando. Entretanto, da original para a versão profunda essa proporção cai bastante. Isso mostra que a proporção de não diferenciabilidade depende fortemente do número de camadas da rede, mas depende pouco do número bruto de parâmetros. Além disso, essa relação não parece ser proporcional, uma vez que, embora o número de camadas convolucionais apenas dobrou, a proporção é 17 vezes menor.

Comparando a rede original com a rede dobrada, também é possível notar que o que faz a proporção de pontos de não diferenciabilidade das duas ser similar é o fato de que tanto o valor de chamadas ao gradiente num ponto de não diferenciabilidade quanto o de gradientes totais terem praticamente dobrado, o que mantém a proporção similar. Assim, parece que o valor bruto de ocorrência de gradientes em pontos de não diferenciabilidade é proporcional ao número de neurônios/filtros, uma vez que da rede original para a dobrada a diferença é que o número de filtros de convolução está dobrado.

5.4 Resultados na MobileNetV3

As análises realizadas para a rede Hochuli são repetidas com a MobileNetV3. Quanto à proporção de não diferenciabilidade, a rede tem 57.792.000.000 chamadas de gradiente das suas funções de ativação, dos quais 2.580 são em pontos de não diferenciabilidade. Isso representa uma proporção de 1 para 22.403.473. Essa proporção é 171 vezes menor que a rede de Hochuli original, e considerando que a MobileNetV3 tem 15 camadas, mais uma vez isso mostra que redes mais profundas tendem a ter proporções de pontos de não diferenciabilidade bem menores. Entretanto, as arquiteturas da MobileNetV3 e da rede de Hochuli são bem diferentes, logo é importante considerar que outros fatores podem ter afetado esse resultado. Por exemplo, a MobileNetV3, além da ReLU, usa outras duas funções de ativação, a Hardswish e a Hardsigmoid, que possuem dois pontos de não diferenciabilidade cada, o que por si só aumenta a probabilidade de cair nesses pontos.

Tabela 4: Métricas dos modelos com rede MobileNetV3 na melhor época.

Métrica	Não Dif.	Dif.
Melhor Época	14	11 (-21,43%)
Acurácia (teste)	88,15%	88,05% (-0.12%)
Tempo Médio por Época (s)	60,41	57,02 (-5,61%)
Tempo Médio de Previsão (s)	$1,79 \cdot 10^{-4}$	$2,01 \cdot 10^{-4}$ (12,84%)
Gradientes Não Dif.	2.580	-
Gradientes Totais	$5,78 \cdot 10^{10}$	-
Proporção de Não Dif.	$4,46 \cdot 10^{-8}$	-

A Figura 9 mostra o valor da função de perda nos dados de validação nos modelos da MobileNetV3. Assim como na rede de Hochuli, o valor dos dois modelos é parecido ao longo do treino, embora o modelo não diferenciável tenha perda consistentemente menor ao longo do treino. Também é possível notar que os modelos convergem já no início do treino, devido ao uso da rede pré-treinada.

A Tabela 4 compara os modelos nas métricas de interesse. Os resultados são similares aos da rede de Hochuli: o modelo diferenciável atinge sua melhor época antes, e os modelos obtêm acurácia de teste parecida, com o modelo não diferenciável realizando classificações mais rapidamente.

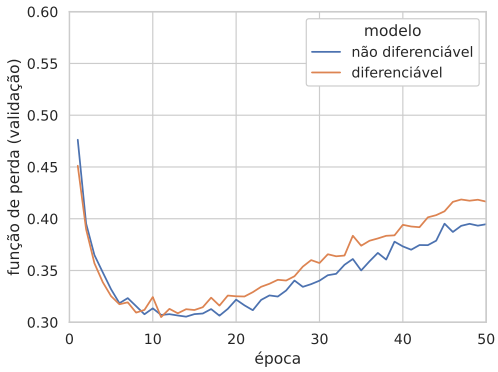

Figura 9: Função de perda na validação da MobileNet V3 por época.

6 CONCLUSÃO

Respondendo às perguntas de pesquisa 1 e 3, os experimentos mostram que pontos sem derivada definida em funções de ativação não diferenciáveis surgem raramente durante o treinamento de modelos de aprendizado profundo, e redes com maior número de camadas tendem a ter proporções de gradientes em pontos de não diferenciabilidade extremamente menores do que redes mais rasas. O número bruto de parâmetros da rede parece afetar pouco essa proporção, apenas aumentando a quantidade bruta de gradientes não bem definidos proporcionalmente ao número de neurônios/filtros.

Quanto à pergunta de pesquisa 2, podemos notar que gradientes em pontos não definidos levam a rede a demorar mais épocas para convergir durante o treinamento, porém não causa impacto real na acurácia das previsões da rede. Além disso, por geralmente serem mais simples de calcular, funções de ativação não diferenciáveis tendem a ajudar o modelo a fazer previsões mais rápidas, embora a diferença seja pequena.

Este trabalho considerou a tarefa de classificação de imagens de dimensões pequenas como forma de realizar experimentos. Entretanto, futuros trabalhos podem ser produzidos com tarefas e arquiteturas diferentes para avaliar se as conclusões se mantêm. Por exemplo, um estudo pode ser feito sobre a tarefa de Processamento de Linguagem Natural (PLN) com *Transformers* [25] para avaliar os resultados.

AGRADECIMENTOS

Esse trabalho foi apoiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) – Projeto 405511/2022-1.

REFERÊNCIAS

[1] Ernst Kussul and Tatiana Baidyk. Improved method of handwritten digit recognition tested on mnist database. *Image and Vision Computing*, 22(12):971–981, 2004.

[2] Audrey Chen. Deep learning in real estate prediction: An empirical study on california house prices. 2024.

[3] Swasti Jain, Sonali Dash, Rajesh Deorari, et al. Object detection using coco dataset. In *2022 International Conference on Cyber Resilience (ICCR)*, pages 1–4. IEEE, 2022.

[4] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[5] C.M. Bishop and H. Bishop. *Deep Learning: Foundations and Concepts*. Springer International Publishing, 2023. ISBN 9783031454684.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.

[8] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[9] David Bertoin, Jérôme Bolte, Sébastien Gerchinovitz, and Edouard Pauwels. Numerical influence of relu'(0) on backpropagation. *Advances in Neural Information Processing Systems*, 34:468–479, 2021.

[10] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[11] Joe Kilian and Hava T Siegelmann. On the power of sigmoid neural networks. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 137–143, 1993.

[12] Muye Chen et al. Vanishing gradient problem in training neural networks. 2022.

[13] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[14] Chih-Hsiang Chang, Hsu-Yu Kao, and Shih-Hsu Huang. Hardware implementation for multiple activation functions. In *2019 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pages 1–2. Ieee, 2019.

[15] Minhyeok Lee. Mathematical analysis and performance evaluation of the gelu activation function in deep learning. *Journal of Mathematics*, 2023(1):4229924, 2023.

[16] Ang Li, Haolin Wu, Yizhuo Wu, Qinyu Chen, Leo CN de Vreede, and Chang Gao. Dpd-neuralengine: A 22-nm 6.6-tops/w/mm recurrent neural network accelerator for wideband power amplifier digital pre-distortion. *arXiv preprint arXiv:2410.11766*, 2024.

[17] Sai Abhinav Pydimarry, Shekhar Madhav Khairnar, Sofia Garces Palacios, Ganesh Sankaranarayanan, Darian Hoagland, Dmitry Nepomnayshy, and Huu Phong Nguyen. Evaluating model performance with hard-swish activation function adjustments. *arXiv e-prints*, pages arXiv–2410, 2024.

[18] R Tyrrell Rockafellar and Roger J-B Wets. *Variational analysis*, volume 317. Springer Science & Business Media, 2009.

[19] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Andre G Hochuli, Alceu S Britto, Paulo RL de Almeida, Williams BS Alves, and Fábio MC Cagni. Evaluation of different annotation strategies for deployment of parking spaces classification systems. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[22] Paulo RL De Almeida, Luiz S Oliveira, Alceu S Britto Jr, Eunelson J Silva Jr, and Alessandro L Koerich. Pklot–a robust dataset for parking lot classification. *Expert Systems with Applications*, 42(11):4937–4949, 2015.

[23] Brett Koonce and Brett Koonce. Mobilenetv3. *Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization*, pages 125–144, 2021.

[24] Autar K Kaw, Egwu K Kalu, and Duc Nguyen. *Numerical methods with applications*. University of South Florida, 2009.

[25] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.