

UNIVERSIDADE FEDERAL FLUMINENSE

EDUARDO VERA SOUSA

**SOBRE O MERGULHO CONFORMAL DE
REDES NEURAIS CONVOLUCIONAIS
SEQUENCIAIS**

NITERÓI

2022

UNIVERSIDADE FEDERAL FLUMINENSE

EDUARDO VERA SOUSA

SOBRE O MERGULHO CONFORMAL DE REDES NEURAIS CONVOLUCIONAIS SEQUENCIAIS

Tese apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito para a obtenção do Grau de Doutor em Computação. Área de concentração: Ciência da Computação

Orientador:

LEANDRO A. F. FERNANDES

Co-orientadora:

CRISTINA NADER VASCONCELOS

NITERÓI

2022

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

S725s Sousa, Eduardo Vera
 Sobre o Mergulho Conformal de Redes Neurais Convolucionais
Sequenciais / Eduardo Vera Sousa ; Leandro Augusto Frata
Fernandes, orientador ; Cristina Nader Vasconcelos,
coorientadora. Niterói, 2022.
 93 p. : il.

 Tese (doutorado)-Universidade Federal Fluminense, Niterói,
2022.

DOI: <http://dx.doi.org/10.22409/PGC.2022.d.05079507390>

 1. Redes neurais. 2. Função de ativação. 3.
Associatividade. 4. Custo computacional. 5. Produção
intelectual. I. Fernandes, Leandro Augusto Frata, orientador.
II. Vasconcelos, Cristina Nader, coorientadora. III.
Universidade Federal Fluminense. Instituto de Computação.
IV. Título.

CDD -

EDUARDO VERA SOUSA

SOBRE O MERGULHO CONFORMAL DE REDES NEURAIIS CONVOLUCIONAIS
SEQUENCIAIS

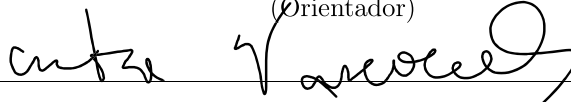
Tese apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito para a obtenção do Grau de Doutor em Computação. Área de concentração: Ciência da Computação

Aprovada em Janeiro de 2022.

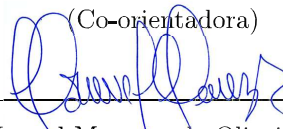
BANCA EXAMINADORA



Prof. D.Sc. Leandro Augusto Frata Fernandes - IC - UFF
(Orientador)



Profa. D.Sc. Cristina Nader Vasconcelos - IC - UFF
(Co-orientadora)



Prof. D.Sc. Manuel Menezes de Oliveira Neto - UFRGS



Prof. D.Sc. Sandra Avila - IC - UNICAMP



Profa. D.Sc. Aline Marins Paes - IC - UFF



Prof. D.Sc. Lúcia M. A. Drummond - IC - UFF

Niterói

2022

À Savya, Cléo e Nonato.

Resumo

Redes Neurais Convolucionais ou CNNs (do inglês *Convolutional Neural Networks*) fazem parte de um amplo ferramental de aprendizado de máquina. Esse tipo de abordagem, cujos modelos têm evoluído continuamente ao longo da última década, possui aplicações nos mais variados contextos, indo de carros autônomos à detecção precoce de doenças. À medida em que o número de camadas de uma rede neural convolucional aumenta, a rede se torna capaz de aprender de maneira mais robusta, sobretudo quando se trata de dados mais complexos. Esse aumento na profundidade das redes, no entanto, tende a impactar diretamente no custo computacional dos processos de treinamento e inferência.

As arquiteturas de redes neurais convolucionais tradicionalmente utilizam funções não-lineares como funções de ativação visando o aprendizado das relações não-lineares entre as características (*features*) presentes nos dados. Essas funções costumam ser aplicadas utilizando-se, como entrada, cada elemento do tensor que representa o dado processado pela camada. Essa forma de utilização acaba por se mostrar uma restrição ao uso de propriedades como a associatividade, por exemplo. Essa propriedade em particular permitiria a representação de redes neurais convolucionais como uma composição compacta das operações executadas pelas camadas da rede. A grande vantagem dessa composição vem da possibilidade de redução do número de operações necessárias à inferência uma vez que a rede esteja treinada, tornando o processo mais rápido e independente do número de camadas da rede.

Nesse trabalho é apresentada a ReSPro, uma função de ativação diferenciável modelada no espaço conforme de geometria e totalmente linear apesar de sua interpretação geométrica não-linear. Na prática, a ReSPro é definida através de um processo de reflexão numa hiper-esfera, seguida por uma escala uniforme e uma projeção num espaço-base. Uma característica dessa função é a possibilidade de representá-la como uma composição de tensores, sendo a associatividade uma das propriedades mais interessantes dessa composição. Utilizando-se de modificações nas estruturas das camadas de uma rede neural convolucional, foi possível redesenhá-las como camadas chamadas ConformalLayers. Essas camadas, já mergulhadas no modelo conforme, também possuem a propriedade da associatividade e, juntamente com a função ReSPro, permitem a modelagem de redes neurais convolucionais como uma composição compacta de tensores esparsos. Essa representação apresenta uma série de vantagens, dentre elas uma redução no consumo de memória em comparação com arquiteturas não-associativas, além de inferência do modelo em tempo constante, de maneira independente da quantidade de camadas da rede neural e relacionado apenas ao tamanho das entradas e saídas da rede.

Palavras-chave: redes neurais, função de ativação, linear, tensor, custo computacional

Abstract

Convolutional Neural Networks or CNNs are part of a broader machine learning tool. This type of approach, whose models have continuously evolved over the last decade, has multiple applications in the most varied contexts, ranging from autonomous cars to early detection of diseases. As the number of layers of a convolutional neural network increases, the network learning becomes more robust, especially when dealing with more complex data. This depth increase of networks, however, tends to impact the computational cost of training and inference processes.

Convolutional neural network architectures traditionally use nonlinear functions as activation functions enables learning the nonlinear relationships between features present in the data. These functions are usually applied using each tensor element that represents the data processed by the layer as input. This form of use turns out to be a constraint for the properties usage such as associativity, for example. This particular property would allow the representation of convolutional neural networks as a compact composition of the operations performed by the network. The great advantage of this composition comes from the possibility of reducing the number of calculations needed for inference once the network has done training, making the process faster and independent of the number of layers in the network.

In this work, we present ReSPro: a linear and differentiable activation function modeled in the conformal space of geometry, despite its non-linear geometric interpretation. In practical terms, ReSPro is defined through a process of reflection on a hypersphere, followed by a uniform scale and a projection onto a base space. A feature of this function is the possibility of representing it as a composition of tensors, associativity being one of the most attractive properties of this composition. By modifying the layer structures of a convolutional neural network, it was possible to redesign them as layers called ConformalLayers. These layers, embedded in the conformal model, are also associative and enable the modeling of convolutional neural networks as a compact composition of sparse tensors. This representation provides several advantages, including a reduction in memory footprint compared to non-associative architectures, in addition to constant-time model inference, regardless of the number of layers of the neural network and related only to the size of the network inputs and outputs.

Keywords: neural networks, activation function, linear, tensor, computational cost

Lista de Figuras

2.1	Modelo de neurônio artificial que representa a operação OR	7
2.2	Modelo de neurônio artificial que representa a operação AND	7
2.3	Estrutura básica de uma rede neural	8
2.4	Arquitetura de uma rede neural convolucional sequencial genérica	10
2.5	Funções de ativação	11
2.6	Derivadas das funções de ativação	12
2.7	Convolução aplicada sobre imagem para a extração de características	14
2.8	Convolução aplicada sobre imagem para a reamostragem de pixels	16
2.9	Operação de dualização	19
2.10	Reflexão de um vetor a em um blade $M_{\langle n-1 \rangle}$	22
2.11	Representação de um círculo no modelo conforme 2-D	24
4.1	Passo a passo da intuição para a formulação da ReSPro	34
4.2	Representação gráfica do mapeamento produzido pela ReSPro	35
4.3	A função <code>forward</code> do módulo <code>cl.ConformalLayers</code>	49
5.1	Implementação e arquitetura da rede <code>BaseLinearNet</code>	56
5.2	Implementação e arquitetura da rede <code>BaseReLUNet</code>	56
5.3	Implementação e arquitetura da rede <code>BaseReSProNet</code>	56
5.4	Implementação da arquitetura <code>LeNetCL</code>	60
5.5	Implementação da arquitetura <code>LeNet</code>	60
5.6	Arquiteturas <code>LeNetCL</code> e <code>LeNet</code>	60
5.7	Implementação da arquitetura <code>DkNetCL</code>	62
5.8	Implementação da arquitetura <code>DkNet</code>	62

5.9	Arquitetura das redes DkNetCL e DkNet	62
5.10	Tempos de inferência para as arquiteturas DkNetCL e DkNet	63
5.11	Implementação da arquitetura D3ModNetCL	65
5.12	Implementação da arquitetura D3ModNet	65
5.13	Arquitetura das redes D3ModNetCL e D3ModNet	65
5.14	Tempo de inferência para os modelos D3ModNetCL e D3ModNet	66
5.15	Consumo de memória da GPU durante o processo de inferência das arquiteturas D3ModNetCL e D3ModNet	66
A.1	Hiperparâmetros que levam às top-10 acurácias para a base MNIST	80
A.2	Hiperparâmetros que levam às top-10 acurácias para a base FashionMNIST	81
A.3	Hiperparâmetros que levam às top-10 acurácias para a base CIFAR10 . . .	82

Sumário

1	Introdução	1
1.1	Problema de Pesquisa	2
1.2	Objetivos	3
1.3	Ideia Central	3
1.3.1	Questões de Pesquisa	4
1.4	Contribuições e Demonstração	4
2	Fundamentação Teórica	6
2.1	Visão Geral sobre Redes Neurais	6
2.1.1	Neurônios Artificiais e <i>Perceptrons</i>	6
2.2	Camadas de Redes Neurais Convolucionais	10
2.2.1	Funções de Ativação	11
2.2.2	Convolução	14
2.2.3	Reamostragem	15
2.3	Conceitos de Álgebra Geométrica	16
2.3.1	Espaço Multivetorial	16
2.3.2	Produto Externo	17
2.3.3	Contração à Esquerda	18
2.3.4	Dualização	19
2.3.5	Produto Geométrico	21
2.3.6	Modelos de Geometria	23
2.4	Discussão	25

3	Trabalhos Relacionados	27
3.1	Compressão de Redes Neurais	27
3.2	Redes Neurais Computacionalmente Eficientes	28
3.3	Aplicações de Álgebra Geométrica em Redes Neurais	30
3.4	Discussão	31
4	ConformalLayers	33
4.1	ReSPro: de Transformações a Funções de Ativação	33
4.2	ReSPro: de Funções de Ativação a Tensores	36
4.3	De Camadas Lineares a ConformalLayers em Redes Neurais Convolucionais	42
4.4	Abordagem Associativa para Redes Neurais Convolucionais	45
4.5	Implementação das ConformalLayers	48
4.6	Discussão	51
5	Experimentos e Resultados	53
5.1	Experimento I - <i>Baseline</i> Linear	54
5.2	Experimento II - LeNet \times LeNetCL	59
5.3	Experimento III - Profundidade da Rede \times Tempo de Inferência	61
5.4	Experimento IV - Tamanho do Lote \times Tempo de Inferência	64
5.5	Discussão	68
6	Conclusões e Trabalhos Futuros	70
6.1	Trabalhos Futuros	71
6.1.1	Ajuste mais Preciso do Parâmetro α	71
6.1.2	Representação de Camadas Totalmente Conectadas	71
6.1.3	Modelagem do viés em Camadas Convolucionais	72
6.1.4	Aplicação das ConformalLayers no processo de treinamento	72

6.1.5	ConformalLayers em outras arquiteturas de redes e tarefas de aprendizado de máquina	72
6.1.6	Análise do desempenho das ConformalLayers em outras arquiteturas de GPU	73
	Referências	74
	Apêndice A – Seleção de Hiperparâmetros	79

Capítulo 1

Introdução

Redes neurais são um ferramental de aprendizado de máquina que visa modelar o processo de aprendizado de maneira similar ao que ocorre no cérebro de animais [1]. Essa é uma área com grande avanço recente devido, sobretudo, ao aumento do poder computacional, essencial aos processos de treinamento e inferência das redes. De modo geral, redes neurais podem ser definidas como conjuntos de funções acopladas, nesse contexto comumente representadas por neurônios artificiais, que visam aproximar uma função que mapeia o conjunto de entradas da rede para o conjunto de saídas da rede [2].

O treinamento de redes neurais é um processo iterativo dito supervisionado quando é baseado na apresentação de pares de entradas e saídas conhecidas (*i.e.*, anotadas) à rede e no ajuste dos pesos associados a cada entrada. Na prática, esse tipo de aprendizado é um processo de otimização cujo objetivo é minimizar uma função de erro associado à diferença entre a saída conhecida e a saída obtida. Uma das maneiras de treinar essas redes é através da retro propagação do erro e nos ajustes dos pesos, utilizando as derivadas parciais dos erros em relação aos pesos. Esse processo em particular é detalhado no Capítulo 2.

No contexto de visão computacional existem diversas aplicações para redes neurais, desde segmentação [3, 4] a reconhecimento semântico de ações e cenas [5, 6]. Considerando a estrutura dos sinais utilizados como entrada (áudios, imagens etc.), as primeiras camadas da rede normalmente são baseadas em convoluções para a extração das características (*features*) mais relevantes para a tarefa. Nesse contexto, a convolução é uma operação linear fundamental para tarefas relacionadas à visão computacional posto que, dependendo do filtro utilizado (também chamado de *kernel*), é capaz de realçar características de interesse em um sinal (*e.g.*, regiões de bordas, regiões de baixa frequência etc.) ou ainda extrair correlações entre os componentes do sinal. Assim, numa rede neural convolucional voltada à classificação de imagens, por exemplo, as camadas convolucionais selecionam

as características da imagem mais relevantes para a classificação e, dependendo da arquitetura utilizada, as entrega para as camadas totalmente conectadas, responsáveis pela distinção entre as várias características, até chegar às saídas. É possível também criar redes totalmente convolucionais, em que a porção convolucional é responsável pela extração e a distinção das características. É importante enfatizar que, em ambos os casos, a retropropagação compreende também as camadas convolucionais. Isso garante que o processo de extração de características seja otimizado ao longo do processo de treinamento, *i.e.*, as características mais relevantes para o domínio são extraídas dos dados de entrada.

Um dos componentes essenciais de redes neurais é a função de ativação. Sua interpretação, no que diz respeito à semelhança biológica, é o limiar de excitação de um neurônio, *i.e.*, o quanto a entrada foi capaz de estimular o neurônio a ponto de ele ser ativado [2]. Funções de ativação são normalmente funções não-lineares, algumas com maior semelhança biológica que outras. Um compilado das funções mais utilizadas na literatura, bem como suas características e aplicações é apresentado no Capítulo 2. É importante enfatizar que funções lineares também podem ser utilizadas. A principal consequência, no entanto, é que o comportamento da rede, por construção, se degrada para um comportamento similar ao de um regressor linear perdendo, assim, seu potencial de ajuste ao comportamento não-linear dos dados. Pode-se dizer, assim, que *a utilização de funções não-lineares como funções de ativação permite ao modelo aprender relações não-lineares entre as características presentes nos dados.*

1.1 Problema de Pesquisa

Nas arquiteturas atuais, costuma-se representar os dados de entrada e os dados produzidos por cada camada como tensores. As camadas, por sua vez, processam os dados recebidos de acordo com suas finalidades e considerando no processamento os pesos aprendidos durante o treinamento, quando é o caso. Funções de ativação são tipicamente aplicadas sobre cada elemento dos tensores de entrada dessas funções. A sequência de camadas em uma rede pode ser formada por operações (*e.g.* convolução, reamostragem, ativação) que não podem ser compostas por associatividade. Nesse contexto, o termo associatividade se refere à propriedade matemática da operação binária \circ em um conjunto \mathcal{S} de modo que

$$(x \circ y) \circ z = x \circ (y \circ z) \text{ para todo } x, y, z \in \mathcal{S}. \quad (1.1)$$

Se a sequência for apenas de camadas de convolução, essa sequência pode ser combinada em uma só operação de convolução, pois esta operação é associativa. Entretanto, funções

de ativação e outras operações que não são associativas à convolução impedem a composição das camadas como uma única operação. A impossibilidade de composição associativa entre as operações que compõem uma rede acaba por demandar o armazenamento dos tensores intermediários produzidos por cada camada e um maior número de operações, visto que a simplificação de operações em camadas consecutivas não ocorre.

Seria interessante, assim, (i) poder definir arquiteturas compostas por operações associativas, visando uma representação compacta da rede pela composição de camadas consecutivas, de modo que a quantidade de operações e uso de memória sejam reduzidos após a composição; e (ii) mapear o processo para um domínio linear, inclusive as funções de ativação, de modo que seja possível a representação dessas operações como matrizes ou tensores, porém sem perda da capacidade de modelagem de dados complexos. Esse mapeamento de funções para um domínio linear ao passo em que mantém características não-lineares já é um processo conhecido em álgebra geométrica [7]. Uma visão geral dessa álgebra e a forma como é aplicada na formulação apresentada nesse trabalho através da geometria conforme são descritos nos Capítulos 2 e 4, respectivamente. A importância da associatividade vem, sobretudo, do fato de ser uma operação natural do domínio de representação tensorial em que as implementações de redes neurais já utilizam. Assim, permitir essa operação e modelar todo o conjunto de camadas como tensores parece ser um caminho natural para essas arquiteturas.

1.2 Objetivos

Esse trabalho possui os seguintes **objetivos gerais**:

G1. Obter uma função que possa ser utilizada como função de ativação para redes neurais, devendo ser associativa e que não comprometa a capacidade de modelagem de relações não-lineares entre as características dos dados;

G2. Tirar proveito da função de ativação modelada em **G1** para definir uma arquitetura de rede neural sequencial em que as camadas façam uso da propriedade associativa.

1.3 Ideia Central

A ideia central desse trabalho pode ser descrita como:

É possível aproveitar operações originalmente descritas em geometria con-

forme, devidamente simplificadas para uma representação com geometria projetiva, para manter todo o processo de treinamento e inferência de redes neurais convolucionais no espectro linear, ainda que a interpretação geométrica das representações seja não-linear e, dessa forma, utilizar a associatividade para compor operações e reduzir consideravelmente o custo associado à inferência em redes neurais convolucionais.

1.3.1 Questões de Pesquisa

A ideia central apresentada levanta algumas questões que atuam como guia para esse trabalho de pesquisa. O Capítulo 5 apresenta experimentos que visam respondê-las.

Q1. Como a utilização de uma função de ativação linear com interpretação geométrica não-linear em redes neurais convolucionais sequenciais, *i.e.*, sem ramificações baseadas em conexões entre camadas não-adjacentes, se mostra melhor quando comparada a redes neurais construídas apenas com funções de ativação lineares?

Q2. Como a composição de uma rede neural convolucional sequencial totalmente associativa se mostra melhor quando comparada com arquiteturas tradicionais, sob a ótica da acurácia de classificação?

Q3. Redes neurais construídas de maneira associativa apresentam algum benefício prático no que diz respeito ao número de operações executadas ou, de maneira mais específica, ao tempo de inferência?

Q4. Aproveitar a associatividade entre as camadas de uma rede neural convolucional sequencial apresenta algum benefício do ponto de vista de utilização de recursos computacionais?

1.4 Contribuições e Demonstração

Uma das contribuições desse trabalho é uma função de ativação totalmente representável como uma composição de tensores, chamada ReSPro, acrônimo para Reflexão (*Reflection*), Escala (*Scaling*) e Projeção (*Projection*), as transformações aplicadas nessa função de ativação. Uma segunda contribuição, decorrente da primeira, vem do fato de que a utilização dessa função de ativação possibilitou a remodelagem da representação de redes neurais sequenciais de modo que suas camadas possam ser combinadas também como uma composição de tensores, chamada ConformalLayers, o que levou a uma redução sig-

nificativa de memória e custo computacional do processo de inferência em redes neurais convolucionais quando comparadas aos modelos tradicionais.

Ao longo do Capítulo 4 serão apresentados mecanismos que permitem a modelagem de redes neurais sequenciais de maneira completamente linear e associativa. De modo geral, a modelagem proposta é que a saída da k -ésima camada da rede neural seja dada por

$$Y^{(k)} = \left(L_M^{(k)} + L_T^{(k)} X \right) X, \quad (1.2)$$

onde X corresponde ao vetor contendo os dados de entrada, $L_M^{(k)}$ é um tensor esparsos de ordem 2 e $L_T^{(k)}$ é um tensor esparsos de ordem 3 que codificam todas as operações executadas entre a primeira e a k -ésima camadas.

As demonstrações e análises, feitas no Capítulo 5, são baseadas em uma implementação das formulações descritas ao longo desse trabalho e são orientados às questões de pesquisa levantadas na Seção 1.3.1. São feitas comparações com redes com apenas uma camada, sendo um *baseline* puramente linear e uma rede que utiliza a função ReLU como ativação. Além disso, é feita uma comparação entre arquiteturas baseadas na LeNet, sendo uma implementada com a função de ativação ReLU e, portanto, não-associativa e uma utilizando-se as ConformalLayers. Adicionalmente, são avaliados os tempos de inferência para as arquiteturas quando variada a profundidade da rede e os tamanhos de batch. Os resultados obtidos apresentam, dentre outros ganhos, redução no consumo de memória e tempo constante de processamento, independente da quantidade de camadas da rede.

Capítulo 2

Fundamentação Teórica

Nesse capítulo serão abordados conceitos fundamentais para uma melhor compreensão da abordagem desenvolvida. Ao passo em que a Seção 2.1 aborda noções básicas de redes neurais e redes neurais profundas, a Seção 2.2 estende os conceitos de redes neurais ao explorar redes neurais convolucionais e analisar a construção e a semântica dos tipos de camadas mais comumente utilizados na literatura. Por fim, a Seção 2.3 descreve conceitos e operações de álgebra geométrica e espaços métricos, essenciais para a plena compreensão da técnica apresentada.

2.1 Visão Geral sobre Redes Neurais

Nessa seção será abordado cada componente necessário à topologia de uma rede neural básica, concluindo com a forma como esses componentes interagem entre si para dar forma à rede propriamente dita.

2.1.1 Neurônios Artificiais e *Perceptrons*

O neurônio artificial, originalmente modelado por McCulloch e Pits [8], foi o primeiro modelo matemático para o comportamento de um neurônio, dado por

$$\hat{y} = \sum_i x_i \geq t, \quad (2.1)$$

onde $x_i \in \{0, 1\}$ representa a i -ésima entrada, $\hat{y} \in \{0, 1\}$ representa a resposta do neurônio às entradas e $t \in \mathbb{R}$ é o limiar de ativação. Perceba que a modelagem lida apenas com entradas e saídas binárias, o que a torna, assim, biologicamente imprecisa. Outra desvantagem diz respeito à atribuição de pesos a cada entrada: o modelo de McCulloch e

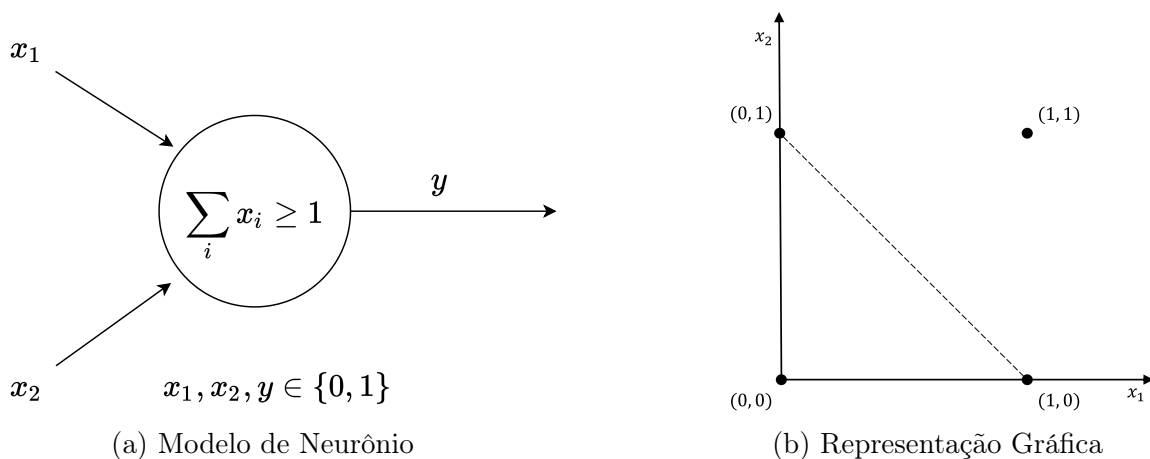


Figura 2.1: Modelo de neurônio artificial que representa a operação **OR**, com sua respectiva interpretação geométrica, em que valores sobre e acima da linha tracejada indicam entradas que ativam o neurônio. Assim, a linha permite separar os valores, *i.e.*, os dados de entrada são linearmente separáveis.

Pits atribui o mesmo peso para todas as entradas, de modo que não existe um conceito de importância de uma entrada em relação às demais. Apesar da abordagem simples, esse modelo representa bem um neurônio que reproduz uma operação lógica, como apresentado nas Figuras 2.1 e 2.2, demonstrando as operações lógicas **or** e **and**, respectivamente, e suas interpretações geométricas, em que $x_1, x_2 \in \{0, 1\}$ correspondem às entradas da função definida no interior do neurônio. A linha tracejada indica a separação linear entre as entradas de modo que, valores sobre e acima dessa linha produzem saídas que ativam os neurônios e o oposto acontece com valores abaixo.

Aqui é possível perceber que o modelo se ajusta bem a entradas linearmente sepa-

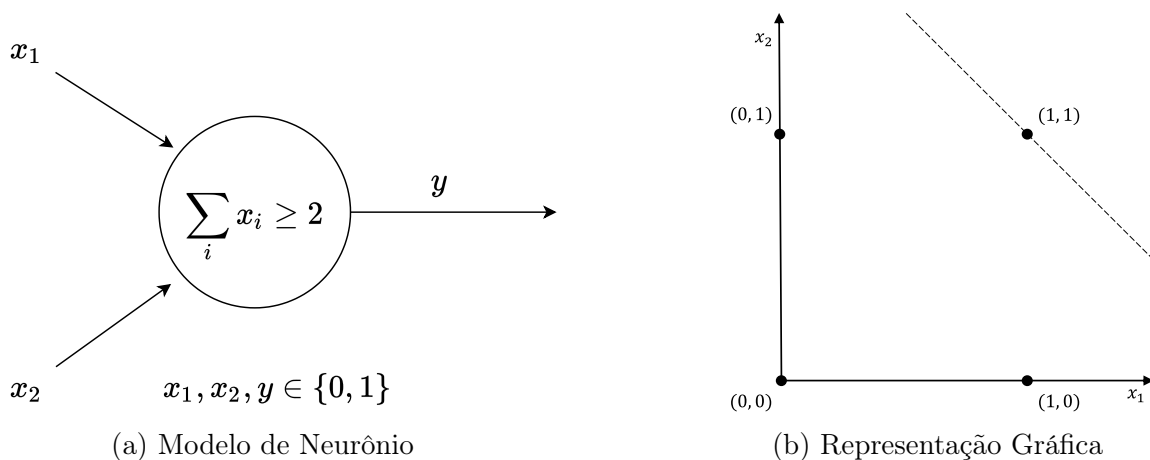


Figura 2.2: Modelo de neurônio artificial que representa a operação **AND**, com sua respectiva interpretação geométrica. Valores sobre ou acima da linha tracejada indicam entradas que ativam o neurônio.

ráveis, *i.e.*, é possível definir um hiperplano (reta, nessa dimensionalidade) que separe bem as entradas que ativam e não ativam o neurônio. Perceba que isso não acontece com a operação **xor**, por exemplo, não contemplada pelo modelo de neurônio artificial de McCulloch e Pits.

O *perceptron*¹, modelo criado por Rosenblatt [9], é um tipo de neurônio artificial voltado para classificação, também considerando entradas linearmente separáveis. O modelo é descrito por

$$\hat{y} = \begin{cases} 1, & \text{se } \sum_i W_i x_i \geq t \\ 0, & \text{c.c.} \end{cases}, \quad (2.2)$$

onde $x_i \in \mathbb{R}$ representa a i -ésima entrada, $\hat{y} \in \{0, 1\}$ representa a resposta do neurônio às entradas e $t \in \mathbb{R}$ é o limiar de ativação. Perceba que nesse modelo acontece a introdução de um termo $W_i \in \mathbb{R}$ que corresponde ao peso associado a cada entrada. Esse modelo permite adicionar, ainda, o conceito de função de ativação, representado por f , mais bem explorado na Seção 2.2.1. No caso do *perceptron*, essa função de ativação corresponde a uma função de Heaviside [10], dada por

$$f(x) = \begin{cases} 1, & \text{se } x > 0 \\ 0, & \text{c.c.} \end{cases}, \quad (2.3)$$

A utilização dessa função de ativação no modelo do *perceptron* serviu como base para

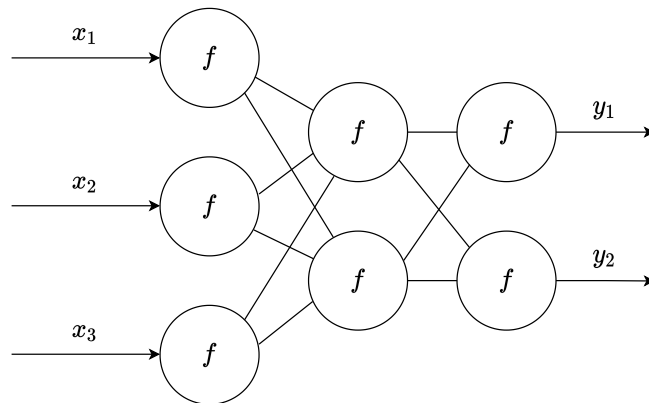


Figura 2.3: Estrutura básica de uma rede neural, nesse caso um *perceptron* multicamadas, em que cada camada pode ser vista como uma composição de várias funções de ativação que utiliza um conjunto de pesos associados a cada neurônio artificial e a função de ativação f para mapear os dados e pesos para um domínio não-linear, conectando as entradas x_i e as saídas y_j .

¹Na literatura é comum referir-se a *perceptron* e neurônio artificial como sinônimos. Na prática, um *perceptron* é um caso especial de neurônio artificial, modelado por Rosenblatt [9], e que utiliza uma função não-linear como função de ativação. Para mais informações, consulte a Seção 2.2.1.

formulações envolvendo outras funções com características similares, mas que implicam numa melhor performance ou que representem melhor o comportamento biológico.

O processo de treinamento do *perceptron* é iterativo. Os pesos W_i são inicializados aleatoriamente e o objetivo é minimizar a função de custo

$$\ell = \min_{W_i} \left| y - f \left(\sum_i W_i x_i + b \right) \right|, \quad (2.4)$$

sendo y a classificação da entrada, já previamente conhecida, f a função de ativação (nesse caso a função de Heaviside), x_i a i -ésima entradas e W_i o peso associado a i -ésima entrada. Os pesos W_i são, na prática, dispostos em tensores, de modo que o produto com todas as entradas possa ser feito de maneira simultânea. Essa é a base das **camadas totalmente conectadas ou camadas densas**. Perceba que agora existe um termo independente b , conhecido como viés (do inglês, *bias*). Esse termo vem da manipulação da primeira parte da Equação 2.2, sendo $b = -t$.

Para a tarefa de classificação, principalmente quando se trata de múltiplas classes, a abordagem mais comum quando se trata de redes neurais é a composição de múltiplos neurônios, onde a saída de uma camada corresponde à entrada da camada seguinte (Figura 2.3). Esse modelo é o *perceptron* multicamadas ou MLP (do inglês, *multi-layer perceptron*).

O treinamento de um MLP é baseado na utilização das derivadas parciais para ajuste dos pesos de cada camada oculta, *i.e.*, camadas na porção interna da rede. De modo geral, a formulação é dada por

$$\delta_j = \begin{cases} \hat{y}_j - y_j, & \text{na camada de saída} \\ f'(\hat{y}_{j-1}) \sum_k \frac{\partial \ell}{\partial \hat{y}_k}, & \text{c.c.} \end{cases}, \quad (2.5)$$

sendo \hat{y}_j a saída estimada na j -ésima camada, \hat{y} a saída anotada na j -ésima camada, f' a derivada da função de ativação, x_j o valor de entrada na j -ésima camada e ℓ a função de erro (Equação 2.4). Perceba que o processo de ajuste dos pesos é recursivo na direção oposta à da rede, processo conhecido como retro-propagação do erro. A cada passo da recursão, $\frac{\partial \ell}{\partial x_i} = \delta_j w_{ij}$ é passado para a camada anterior para o cálculo de δ_{j-1} . Uma vez calculado o valor de cada δ_j , o passo seguinte é a atualização dos pesos, onde $\frac{\partial \ell}{\partial w_{ij}} = \delta_j x_i$. Terminada a retro-propagação do erro, o dado é passado pela rede, num processo chamado propagação à frente, novamente utilizando os pesos reajustados e o processo se repete até um limiar de ℓ aceitável. Levando-se em conta indexação, é possível reorganizar os pesos

e as entradas como matrizes e vetores, o que torna o processo ideal para execução em arquiteturas de processamento paralelo, como GPUs (do inglês, *graphics processing unit*).

2.2 Camadas de Redes Neurais Convolucionais

Segundo Goodfellow *et al.* [2], redes neurais convolucionais são um tipo de rede neural específico para dados organizados em formato de grades. Séries temporais e imagens, por exemplo, são o tipo de dado mais adequado para esse tipo de rede neural, posto que podem ser vistos como grades 1 e 2-dimensionais, respectivamente. Originalmente modelado por LeCun *et al.* [11], redes neurais convolucionais são definidas como redes neurais que utilizam a operação de convolução ao invés do produto de tensores em pelo menos uma de suas camadas. Na prática, a operação executada é a correlação cruzada mas o termo convolução é mais amplamente utilizado na literatura. A principal função dessa operação é a extração das características mais relevantes para a tarefa para a qual a rede neural é modelada.

A porção convolucional possui um processo de treinamento bastante similar à de MLPs comuns, inclusive com o termo de viés (*bias*). A diferença está na operação realizada: ao passo em que camada de MLP é construída com produtos entre os tensores que representam as entradas e os pesos, a camada da porção convolucional é modelada com a correlação cruzada entre os tensores que representam as entradas e os pesos que definem os filtros de convolução. A Figura 2.4 apresenta uma arquitetura de rede neural convolucional sequencial com uma porção convolucional (camadas de 1 a 3) e uma porção totalmente conectada (camadas FC1 a FC3). As componentes mais comuns desse tipo de

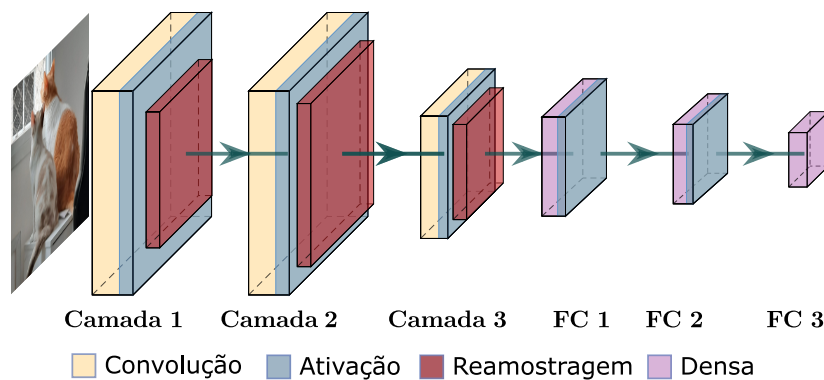


Figura 2.4: Arquitetura de uma rede neural convolucional sequencial genérica. As camadas convolucionais 1, 2 e 3, *i.e.*, o conjunto de camadas antes da porção totalmente conectada F1, F2 e F3 (densas), é responsável pela extração das características do sinal de entrada.

arquiteturas são exploradas ao longo deste Capítulo.

2.2.1 Funções de Ativação

Uma função de ativação f é a função que mapeia a resposta de um neurônio à um estímulo (entrada). Modelos que utilizam funções lineares (Figura 2.5a) como função de ativação se comportam de maneira equivalente à um regressor linear. Tais modelos são, no entanto, limitados no que diz respeito ao aprendizado de comportamentos dos dados. A derivada desse tipo de função 2.6a acaba por se tornar independente da entrada, não adicionando nenhum mapeamento relevante nos dados. Assim, modelos mais robustos de redes neurais utilizam funções de ativação não-lineares, cuja definição varia de acordo com a topologia e com o comportamento dos dados de entrada. É válido notar que boa parte das funções utilizadas como ativação possuem comportamentos similares ao da função de Heaviside, o que demonstra que comportamentos similares a esse são, de fato, importantes para o aprendizado em redes neurais.

A função logística (Figura 2.5c), dada por

$$f(x) = \frac{1}{1 + e^{-x}}, \quad (2.6)$$

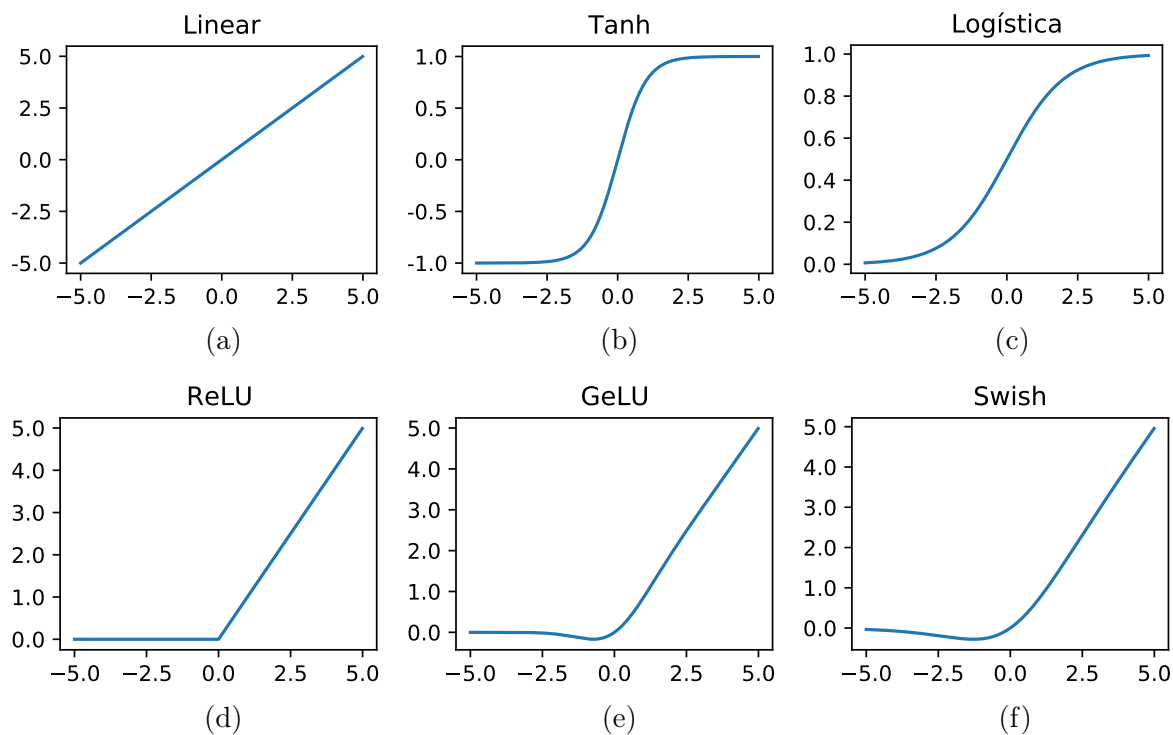


Figura 2.5: Funções de ativação mais comumente utilizadas como funções de ativação em redes neurais convolucionais sequenciais.

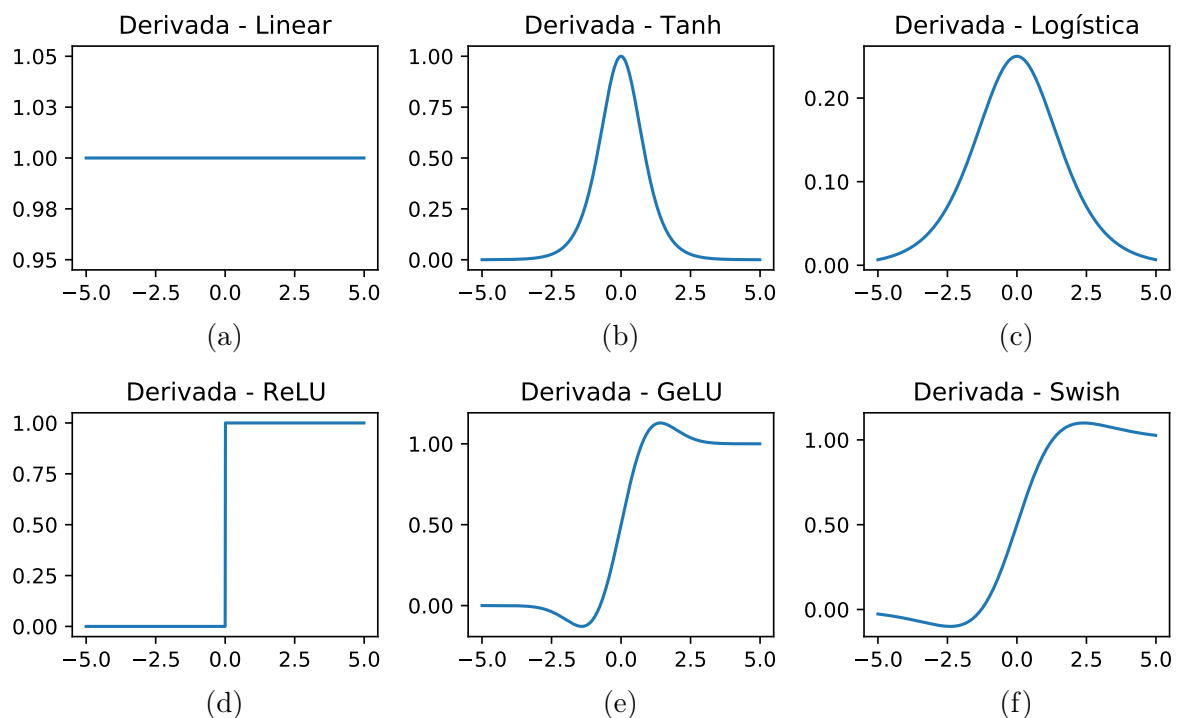


Figura 2.6: Derivadas das funções de ativação apresentadas na Figura 2.5.

é uma função em formato de ‘S’, cuja saída varia entre 0 e 1. Por sua característica mais íngreme, sobretudo no intervalo $-2 < x < 2$, é frequentemente usada em classificadores binários, onde é assumido um limiar de 0,5 para a relação de pertinência ou não a uma classe. Por ser uma função cuja saída não é centrada em zero, o gradiente pode se afastar muito e em direções opostas, aumentando o tempo de convergência. Além disso, essa função pode facilmente saturar, levando ao problema de dissipação do gradiente (do inglês, *vanishing gradient*), onde as derivadas se aproximam tanto de zero que acabam por tornar $\delta_j = 0$ nas camadas intermediárias (ver segunda partição da Equação 2.5) para esse neurônio. Adicionalmente, a função logística possui um tempo de convergência maior em relação aos demais no contexto de redes neurais [2, Capítulo 6].

A utilização da tangente hiperbólica (Figura 2.5b) como função de ativação, dada por

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad (2.7)$$

se mostra mais vantajosa em relação à função logística, apesar da semelhança. Apesar de existir uma relação entre as duas funções, de modo que a função logística pode ser obtida da tangente hiperbólica e vice-versa, a tangente hiperbólica possui intervalo de saída entre -1 e $+1$, de modo que os dados de saída estão centralizados em torno de zero. Isso facilita o aprendizado nas camadas seguintes ao evitar um passo maior que o necessário na descida do gradiente, que passa direto pelo mínimo local que se pretende

alcançar. O problema de dissipação do gradiente, no entanto, se mantém [2, Capítulo 6].

Levando-se em conta o problema recorrente de dissipação do gradiente, foi proposta a função ReLU (do inglês, *rectified linear unit*) [12], representada graficamente na Figura 2.5d. O modelo é dado por

$$f(x) = \max\{0, x\} = \begin{cases} x, & \text{se } x > 0 \\ 0, & \text{c.c.} \end{cases}. \quad (2.8)$$

Esse tipo de função de ativação apresenta três vantagens [2]: (i) elimina o problema de dissipação do gradiente, (ii) é computacionalmente simples. Perceba que, ao analisarmos a função partida, vemos que caso $x > 0$, temos um modelo totalmente linear e, portanto, de baixo custo computacional, *i.e.*, uma vez que a função partida tenha selecionado o neurônio, o processo até a camada seguinte é linear. Ao mesmo tempo, se $x \leq 0$, a função de ativação adiciona esparsidade ao sistema, posto que mapeia as entradas sobre o qual atua para zero. Essa característica pode ser particularmente útil em tarefas não-supervisionadas [2]. Apesar de ser uma das funções de ativação mais utilizadas, a ReLU possui uma limitação que depende dos dados de entrada. Por sua característica de mapear entradas negativas para zero é possível suprimir os valores de uma grande quantidade de neurônios, limitando o aprendizado via retro-propagação de erro para dados de entrada com essa característica [2]. A solução para a atenuação desse problema é a utilização da ReLU paramétrica [13], em que

$$f(x) = \max\{0, x\} + \alpha \min\{0, x\} \quad (2.9)$$

Nesse caso, α é um hiperparâmetro que, assim como os pesos W_i e o viés b da Equação 2.4, também é aprendido ao longo do processo de treinamento. É comum também a utilização da Leaky ReLU [14], em que o valor de α é fixado em 0.1.

A esparsidade dessas funções, como explicitado por Glorot *et al.* [15], aumenta a separabilidade dos dados, *i.e.*, pequenas alterações nos dados de entrada de um *perceptron* impactam menos na representação desses dados, o que auxilia no contexto de generalização do aprendizado das redes.

Em 2016, Hendrycks e Gimpel apresentaram a GeLU [16]: uma função de ativação não linear que modula as componentes do sinal de entrada da rede utilizando a uma distribuição cumulativa de uma gaussiana padrão. O grande diferencial dessa função em relação às demais é o fato dela não ser monotônica, como pode ser visto na Figura 2.5e. Essa função, apesar de mais custosa, é capaz de prover resultados similares ou superiores

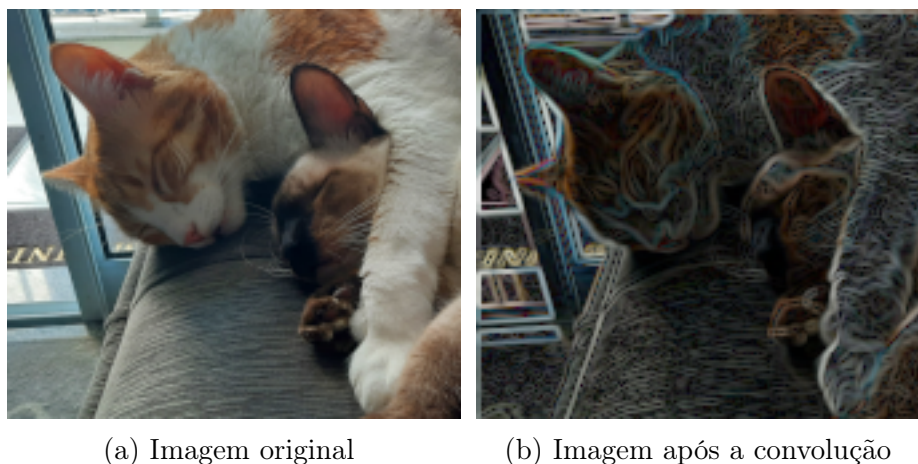


Figura 2.7: Convolução aplicada sobre imagem para a extração de características. A Figura 2.6a corresponde à imagem original. A Figura 2.6b é resultado de uma convolução com filtro de Sobel.

à ReLU. No ano seguinte, em 2017, pesquisadores do Google propuseram a Swish [17] como função de ativação e demonstraram experimentalmente sua performance superior quando comparada às demais. Também não-monotônica (Figura 2.5f), foi demonstrado que redes que utilizam essa função de ativação suportam mais camadas que redes que utilizam ReLU, sem perda de performance no contexto de qualidade do modelo treinado. Também se mostrou mais computacionalmente custosa, o que pode ser verificado por sua definição, dada por

$$f(x) = \frac{x}{1 + e^{-x}}. \quad (2.10)$$

2.2.2 Convolução

A convolução é uma operação básica entre funções em álgebra linear. De modo geral, a operação leva a uma terceira função. Em termos práticos, essa terceira função expressa a interação entre as duas funções operadas. Assim, sejam f e g duas funções contínuas. Sua convolução, representada pelo símbolo $*$ e, em um ponto x , é dada por

$$[f * g](x) = \int_{-\infty}^{+\infty} f(\tau)g(x - \tau)d\tau. \quad (2.11)$$

A convolução de duas funções f e g é dada, portanto, pela soma do produto dessas funções na região em que existe sobreposição entre as funções, levando-se em conta um deslocamento τ . Perceba que o termo $(x - \tau)$ impõe que a função g (ou f , considerando que a convolução é uma operação comutativa) seja espelhada em relação ao eixo das ordenadas.

Computacionalmente, é comum a representação discreta da convolução, dada por

$$[f * g](x) = \sum_{n=-\infty}^{+\infty} f(n)g(x - n). \quad (2.12)$$

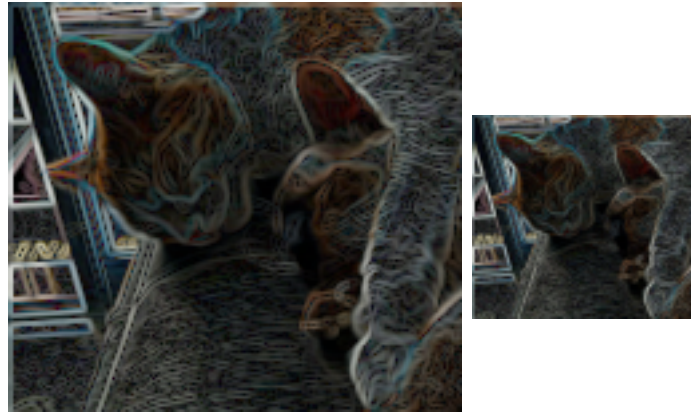
Um das aplicações mais comuns da convolução em Visão Computacional vem da sua capacidade de realçar ou atenuar características em imagens, filtrando porções específicas do sinal. Nesse caso, é comum estender o conceito introduzido na Equação 2.12 para 2-D:

$$[f * g](i, j) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} f(m, n)g(i - m, j - n). \quad (2.13)$$

Para o caso de filtros lineares em imagens, a imagem atua como um dos sinais utilizados como operadores (f ou g). O outro sinal é um sinal específico, o filtro (por vezes também chamado de *kernel*), que depende da tarefa a ser executada. Para detecção de bordas, *i.e.*, regiões de alta frequência na imagem, por exemplo, é comum a utilização do filtro de Sobel [18], ao passo que, para a atenuação dessas mesmas regiões por vezes se utiliza um filtro Gaussiano [19]. A Figura 2.7 demonstra a aplicação de filtros sobre a imagem no intuito de realçar algumas características. A Figura 2.7b representa a aplicação de um filtro de Sobel, voltado à detecção de bordas, sobre a Figura 2.7a.

2.2.3 Reamostragem

As camadas de reamostragem, por sua vez, tem como principal função a redução da carga computacional associada aos processos em redes neurais. Ao diminuir a quantidade de amostras oriundas de determinada região da imagem, as operações posteriores à reamostragem acabam por trabalhar com menos dados, sendo esses mais significativos para o processo de aprendizado, *i.e.*, carregam mais informação associada [20]. Embora diversas abordagens de reamostragem sejam úteis aos processos de treinamento e inferência em redes neurais [20], esta tese explora apenas a reamostragem por média, sobretudo por seu aspecto linear. Isso se dá pelo fato de que é possível modelar a reamostragem por média como uma convolução em que o filtro possui valores constantes iguais a $\frac{1}{s_k}$, sendo s_k o tamanho do filtro. A Figura 2.8b corresponde à Figura 2.8a após a aplicação de um filtro de reamostragem por média. Nesse caso foi utilizado um filtro de tamanho 2×2 com passo igual a 2, *i.e.*, sem sobreposição das regiões de convolução.



(a) Imagem após a convolução (b) Imagem após a reamostragem

Figura 2.8: Convolução aplicada sobre imagem para a reamostragem de pixels. A Figura 2.7a à imagem antes da reamostragem. A Figura 2.8b é o resultado da reamostragem por média utilizando um filtro 2×2 , também aplicada utilizando-se a convolução com um passo igual a 2.

2.3 Conceitos de Álgebra Geométrica

Álgebra geométrica é um ferramental matemático que permite a utilização de entidades geométricas (esferas, pontos, retas, etc) e transformações ortogonais (rotação, translação, escala uniforme, etc) como primitivas a serem operadas diretamente através de operações lineares. A grande vantagem da utilização dessa álgebra no contexto desse trabalho é que, dependendo da métrica, é possível operar primitivas não-lineares n -dimensionais, como hiperesferas, por exemplo, de maneira completamente linear. Nessa Seção são apresentados alguns produtos básicos associados à álgebra geométrica. Referências mais completas podem ser encontradas em [7] e [21].

2.3.1 Espaço Multivetorial

Seja \mathbb{R}^n um espaço vetorial com um conjunto de vetores de base $\{e_i\}_{i=1}^n$, onde $n = p + q + r$, e p , q e r definem a assinatura do espaço métrico, *i.e.*,

$$e_i \cdot e_j = \begin{cases} +1 & , i = j \text{ e } 1 \leq i \leq p, \\ -1 & , i = j \text{ e } p < i \leq p + q, \\ 0 & , i = j \text{ e } p + q < i \leq r, \end{cases} \quad (2.14)$$

onde, \cdot denota o produto interno de vetores.

Perceba que o espaço vetorial \mathbb{R}^n considera apenas primitivas 1-D (*i.e.*, vetores). Em

Tabela 2.1: Espaço multivetorial $\bigwedge \mathbb{R}^4$.

Espaço	Blades de base	Nomenclatura
$\bigwedge^0 \mathbb{R}^4 \equiv \mathbb{R}$	1	Escalar
$\bigwedge^1 \mathbb{R}^4 \equiv \mathbb{R}^4$	e_1, e_2, e_3, e_4	Vetor
$\bigwedge^2 \mathbb{R}^4$	$e_1 \wedge e_2, e_1 \wedge e_3, e_1 \wedge e_4, e_2 \wedge e_3, e_2 \wedge e_4, e_3 \wedge e_4$	2-Vetor
$\bigwedge^3 \mathbb{R}^4$	$e_1 \wedge e_2 \wedge e_3, e_1 \wedge e_2 \wedge e_4, e_1 \wedge e_3 \wedge e_4, e_2 \wedge e_3 \wedge e_4$	Pseudovetor
$\bigwedge^4 \mathbb{R}^4$	$e_1 \wedge e_2 \wedge e_3 \wedge e_4$	Pseudoescalar

álgebra geométrica definimos a partir do espaço vetorial \mathbb{R}^n o espaço multivetorial $\bigwedge \mathbb{R}^n$, onde é possível representar as primitivas de álgebra geométrica como multivetores M . Um multivetor é uma combinação linear de elementos de base de $\bigwedge \mathbb{R}^n$. O conjunto de elementos de base definidos em $\bigwedge \mathbb{R}^4$, por exemplo, é representado na Tabela 2.1. A coluna mais à direita corresponde à denominação particular utilizada para combinações lineares de elementos de base dos espaços 0-dimensional, 1-dimensional, 2-dimensional, ..., $(n-1)$ -dimensional, e n -dimensional. Ao todo, existem 2^n elementos de base no espaço multivetorial $\bigwedge \mathbb{R}^n$. Formalmente, um k -vetor é um multivetor dado pela combinação linear apenas de elementos de base da porção k -vetorial $\bigwedge^k \mathbb{R}^n$ do espaço vetorial $\bigwedge \mathbb{R}^n$.

2.3.2 Produto Externo

O produto externo (denotado por \wedge) é uma operação independente da métrica do espaço e que corresponde ao mapeamento:

$$\wedge : \bigwedge^r \mathbb{R}^n \times \bigwedge^s \mathbb{R}^n \rightarrow \bigwedge^{r+s} \mathbb{R}^n. \quad (2.15)$$

Essa operação possui as seguintes propriedades:

$$\text{antissimetria: } a \wedge b = -b \wedge a, \text{ assim } a \wedge a = b \wedge b = 0$$

$$\text{distributividade: } a \wedge (b + c) = a \wedge b + a \wedge c$$

$$\text{associatividade: } a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

$$\text{comutatividade de escalares: } a \wedge (\alpha b) = \alpha(a \wedge b)$$

Aqui, $a, b, c \in \bigwedge^1 \mathbb{R}^n$ são vetores e $\alpha \in \bigwedge^0 \mathbb{R}^n$ é um valor escalar real.

O produto externo nos permite expandir subespaços lineares (denominados blades na álgebra geométrica) com dimensionalidades mais altas utilizando subespaços lineares com dimensionalidade mais baixa. Logo, o produto externo implementa a ideia de expansão

de subespaços. Por exemplo, o produto externo de dois blades 1-dimensional (*i.e.*, vetores) linearmente independentes define um 2-blade, enquanto o produto externo de três vetores linearmente independentes define um 3-blade, e assim por diante. Formalmente, um k -blade $A_{\langle k \rangle} \in \bigwedge^k \mathbb{R}^n$ é qualquer subespaço linear expandido como o produto externo de k vetores linearmente independentes. Assim, dizemos que o blade $A_{\langle k \rangle}$ possui grau k , onde o termo grau e dimensionalidade são sinônimos. É importante enfatizar que todos os k -blades são também k -vetores, mas nem todos os k -vetores são k -blades. Por exemplo, $e_1 \wedge e_2 + e_3 \wedge e_4$ é um 2-vetor mas não é um 2-blade. No modelo euclidiano, 1-blades podem ser interpretados como linhas retas incluindo a origem de \mathbb{R}^n , 2-blades correspondem a planos, 3-blades a volumes. No modelo homogêneo, frequentemente utilizado em computação gráfica, 1-blades podem ser interpretados como pontos ou direções, 2-blades podem ser interpretados como retas que não necessariamente passam pela origem e assim por diante. No modelo conforme, 1-blades podem ser interpretados como pontos finitos, 2-blades podem ser um par de pontos, sendo um ponto planar caso um dos pontos esteja no infinito, 3-blades podem ser vistos como hipersferas e assim por diante. No modelo de Minkowski (também chamado de modelo espaço-tempo), amplamente empregado na física, sobretudo na formulação da Teoria da Relatividade, 1-blades podem ser interpretados como estados, por exemplo. Algumas dessas métricas são exploradas na Seção 2.3.6.

2.3.3 Contração à Esquerda

Outra operação importante em álgebra geométrica é a contração à esquerda, denotada por \lrcorner . Trata-se um produto métrico que representa remover do operando à direita a porção mais similar ao operando à esquerda, no sentido métrico. Assim, a contração à esquerda consiste num mapeamento:

$$\lrcorner : \bigwedge^r \mathbb{R}^n \times \bigwedge^s \mathbb{R}^n \rightarrow \bigwedge^{s-r} \mathbb{R}^n. \quad (2.16)$$

Dado o exemplo

$$A_{\langle r \rangle} \lrcorner B_{\langle s \rangle} = C_{\langle s-r \rangle}, \quad (2.17)$$

a leitura a ser feita é que $C_{\langle s-r \rangle}$ é a porção $(s-r)$ -dimensional de $B_{\langle s \rangle}$ que é menos parecida com $A_{\langle r \rangle}$ (*i.e.*, com produto interno diferente de zero, ou que é ortogonal a $B_{\langle s \rangle}$). É possível enumerar as seguintes propriedades da contração à esquerda:

$$\text{simetria: } A_{\langle r \rangle} \lrcorner B_{\langle s \rangle} = B_{\langle s \rangle} \lrcorner A_{\langle r \rangle}, \text{ se e somente se } r = s$$

$$\text{distributividade: } A_{\langle r \rangle} \lrcorner (B_{\langle s \rangle} + C_{\langle t \rangle}) = A_{\langle r \rangle} \lrcorner B_{\langle s \rangle} + A_{\langle r \rangle} \lrcorner C_{\langle t \rangle}$$

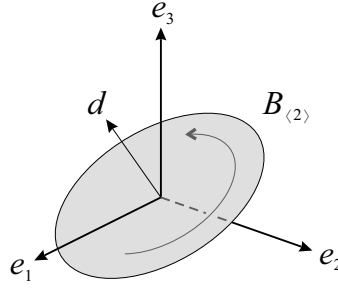


Figura 2.9: Operação de dualização. O vetor d é o dual do plano $B_{(2)}$ em \mathbb{R}^3 sob métrica euclidiana.

comutatividade de escalares: $A_{(r)} \rfloor (\alpha B_{(s)}) = \alpha (A_{(r)} \rfloor B_{(s)})$

Além dessas propriedades, existem algumas relações úteis entre a contração à esquerda e o produto externo, cuja aplicação é mais bem explicada na Seção 2.3.4. A primeira relação é válida para qualquer blade:

$$A_{(r)} \rfloor (B_{(s)} \rfloor C_{(t)}) = (A_{(r)} \wedge B_{(s)}) \rfloor C_{(t)}. \quad (2.18)$$

A segunda relação é:

$$A_{(r)} \rfloor (B_{(s)} \rfloor C_{(t)}) = A_{(r)} \wedge (B_{(s)} \rfloor C_{(t)}), \quad (2.19)$$

e é válida se e somente se $A_{(r)} \subseteq C_{(t)}$.

2.3.4 Dualização

A dualização é a operação denotada por \square^* e define o mapeamento:

$$\square^* : \bigwedge^k \mathbb{R}^n \rightarrow \bigwedge^{n-k} \mathbb{R}^n. \quad (2.20)$$

A interpretação geométrica de $A_{(k)}^*$ é tomar de todo o espaço n -dimensional, *i.e.*, do pseudoscalar unitário $I_{(n)} = e_1 \wedge e_2 \wedge \cdots \wedge e_n$, a porção $(n - k)$ -dimensional que é ortogonal ao blade $A_{(k)}$.

A operação de dualização pode ser expressa utilizando a contração à esquerda:

$$A_{(k)}^* = A_{(k)} \rfloor I_{(n)}^{-1}, \quad (2.21)$$

onde

$$X_{(r)}^{-1} = \frac{1}{\|X_{(r)}\|^2} \tilde{X}_{(r)} \quad (2.22)$$

denota o inverso de $X_{\langle r \rangle}$,

$$\|X_{\langle r \rangle}\|^2 = X_{\langle r \rangle} \rfloor \tilde{X}_{\langle r \rangle} \quad (2.23)$$

é o quadrado da norma reversa de $X_{\langle r \rangle}$, e

$$\tilde{X}_{\langle r \rangle} = (-1)^{\frac{r(r-1)}{2}} X_{\langle r \rangle} \quad (2.24)$$

denota a operação de reversão. A operação apresentada na Equação 2.24 é distributiva sobre a soma. Assim, pode ser aplicada sobre multivetores quaisquer, mesmo que de grau misto, e avaliados com base no grau de cada componente.

Perceba que a contração à esquerda na Equação 2.21 induz a dualização à interpretação geométrica de remover do pseudoescalar tudo o que é similar ao blade $A_{\langle k \rangle}$ dado, mantendo apenas o blade ortogonal a ele. A Figura 2.9 exemplifica o operador de dualização. Essa imagem foi originalmente apresentada em [7]. Nesse exemplo, o pseudoescalar $I_{\langle 3 \rangle}$ corresponde ao espaço 3-dimensional como um todo e $d = B_{\langle 2 \rangle}^*$ é o vetor dual ao 2-blade $B_{\langle 2 \rangle}$.

A dualização é invertível. A operação de desdualização é definida como:

$$A_{\langle k \rangle}^{-*} = A_{\langle k \rangle} \rfloor I_{\langle n \rangle}. \quad (2.25)$$

Tal inversibilidade permite a obtenção do blade original usando a relação definida pela Equação 2.19:

$$(A_{\langle k \rangle}^*)^{-*} = (A_{\langle k \rangle} \rfloor I_{\langle n \rangle}^{-1}) \rfloor I_{\langle n \rangle} = A_{\langle k \rangle} \wedge (I_{\langle n \rangle}^{-1} \rfloor I_{\langle n \rangle}) = A_{\langle k \rangle} \wedge 1 = A_{\langle k \rangle}. \quad (2.26)$$

As Equações 2.18 e 2.19 também ajudam na definição de relações universais entre o produto externo e a contração à esquerda, em termos de dualidade. A primeira relação mostra que o dual do produto externo pode ser substituído pela contração à esquerda:

$$\begin{aligned} (A_{\langle r \rangle} \wedge B_{\langle s \rangle})^* &= (A_{\langle r \rangle} \wedge B_{\langle s \rangle}) \rfloor I_{\langle n \rangle}^{-1} \\ &= A_{\langle r \rangle} \rfloor (B_{\langle s \rangle} \rfloor I_{\langle n \rangle}^{-1}) \\ &= A_{\langle r \rangle} \rfloor B_{\langle s \rangle}^*. \end{aligned} \quad (2.27)$$

A segunda relação mostra como o dual da contração à esquerda e o produto externo são conectados:

$$\begin{aligned} (A_{\langle r \rangle} \rfloor B_{\langle s \rangle})^* &= (A_{\langle r \rangle} \rfloor B_{\langle s \rangle}) \rfloor I_{\langle n \rangle}^{-1} \\ &= A_{\langle r \rangle} \wedge (B_{\langle s \rangle} \rfloor I_{\langle n \rangle}^{-1}) \\ &= A_{\langle r \rangle} \wedge B_{\langle s \rangle}^*. \end{aligned} \quad (2.28)$$

2.3.5 Produto Geométrico

O produto mais importante em álgebra geométrica é o produto geométrico. Na prática, a maioria dos demais produtos podem ser extraídos dele (ver [21] para detalhes). O produto geométrico consiste em um mapeamento

$$\bigwedge \mathbb{R}^n \times \bigwedge \mathbb{R}^n \rightarrow \bigwedge \mathbb{R}^n. \quad (2.29)$$

Suas propriedades são:

$$\text{distributividade: } A(B + C) = AB + AC$$

$$\text{associatividade: } A(BC) = (AB)C$$

$$\text{não-comutatividade no caso geral: } \exists A, B \in \bigwedge \mathbb{R}^n : AB \neq BA$$

Um exemplo do mapeamento descrito na Equação 2.29 envolvendo vetores e multivetores gerais pode ser definido como:

$$aB = a \rfloor B + a \wedge B, \quad (2.30)$$

onde a é um vetor e B é um multivetor. Da Equação 2.30 e das propriedades do produto geométrico, é possível definir o produto geométrico entre quaisquer pares de multivetores. Mais detalhes podem ser vistos em [21].

Perceba na Equação 2.30 que o multivetor resultante do produto geométrico aB possui uma porção métrica ($a \rfloor B$) e uma porção não-métrica ($a \wedge B$). Essa composição torna esse produto invertível, enquanto o operando da direita for, também, invertível, *i.e.*,

$$A / B = AB^{-1}, \quad (2.31)$$

para $\|B\|^2 \neq 0$, onde $/$ denota o inverso do produto geométrico.

Uma das principais aplicações do produto geométrico é na definição de transformações ortogonais, *i.e.*, transformações lineares que preservam o comprimento de vetores e ângulos entre eles, como rotação, transformação, e escala uniforme. Essas transformações podem ser modeladas como uma sequência de reflexões em pseudovetores. Na prática, dado um pseudovetor $M_{\langle n-1 \rangle}$ e seu vetor dual $v = M_{\langle n-1 \rangle}^*$, o vetor refletido a' de um vetor a em $M_{\langle n-1 \rangle}$ pode ser expresso como

$$a' = -vav^{-1}, \quad (2.32)$$

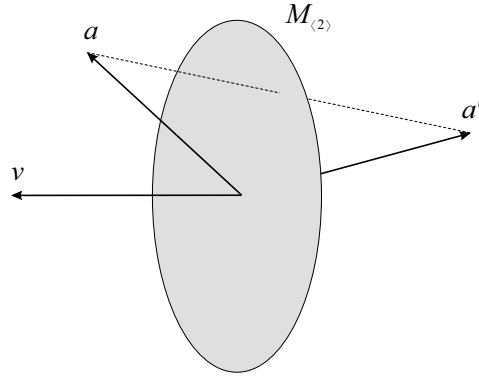


Figura 2.10: Reflexão de um vetor a em um blade $M_{\langle n-1 \rangle}$, geometricamente interpretado como um plano em 3-D passando pela origem de um espaço euclidiano, produzindo um vetor a' . Para a construção dessa reflexão é necessária a utilização do vetor $v = M_{\langle n-1 \rangle}^*$.

onde v^{-1} denota o inverso de v (Equação 2.22). Um exemplo de aplicação da Equação 2.32 é mostrada na Figura 2.10, originalmente disponível em [7].

Perceba que reflexões podem ser combinadas. Assim, é possível aplicar uma reflexão seguida por outra reflexão e assim por diante::

$$a'' = ua'u^{-1} = u(vav^{-1})u^{-1}, \quad (2.33)$$

onde a'' é a reflexão do vetor a' no blade $N_{\langle n-1 \rangle}$ usando seu dual, dado por $u = N_{\langle n-1 \rangle}^*$. Usando a associatividade do produto geométrico, é possível escrever a Equação 2.33 como:

$$a'' = u(vav^{-1})u^{-1} = (uv)a(v^{-1}u^{-1}) = (uv)a(uv)^{-1}. \quad (2.34)$$

Na Equação 2.34, uv é chamado 2-versor. Formalmente, um k -versor \mathcal{V} é definido como o produto geométrico de k vetores invertíveis. Essas primitivas de grau misto codificam transformações ortogonais.

Apesar das Equações 2.32 e 2.33 mostrarem apenas vetores sendo transformados por versores, é importante enfatizar que a construção envolvendo um sanduíche de produtos de versores pode ser aplicado a qualquer multivetor, considerando que o produto geométrico é distributivo sobre a soma e versores preservam a estrutura do produto externo, *e.g.*,

$$\mathcal{V}(a \wedge b)\mathcal{V}^{-1} = (\mathcal{V}a\mathcal{V}^{-1}) \wedge (\mathcal{V}b\mathcal{V}^{-1}), \quad (2.35)$$

onde \mathcal{V} é um versor.

2.3.6 Modelos de Geometria

Nessa Seção, serão explorados brevemente alguns dos modelos de geometria mais utilizados, bem como as transformações que podem ser codificadas como versores e a interpretação geométrica de seus blades.

O **modelo Euclidiano**, como o nome indica, assume métrica euclidiana para o espaço vetorial. Em termos práticos, isso significa que o produto interno entre pares de vetores de base do espaço $\mathbb{R}^{n,0}$ define uma matriz identidade correspondendo, assim, à uma reescrita da Equação 2.14 para $p = n$ e $q = 0$. A matriz de métrica do modelo Euclidiano é dada por

$$\begin{array}{c|cccc} \cdot & e_1 & e_2 & \dots & e_n \\ \hline e_1 & 1 & 0 & \dots & 0 \\ e_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e_n & 0 & 0 & \dots & 1 \end{array}$$

Nesse modelo, o espaço base, onde ocorre a interpretação geométrica, e o espaço de representação, que consiste no espaço vetorial como um todo, correspondem ao mesmo espaço. Isso leva à interpretação direta de vetores como retas, 2-blades como planos, 3-blades como volumes, etc., todos necessariamente incluindo o vetor zero, *i.e.*, a origem do espaço. As únicas transformações codificadas por versores neste modelo são a reflexão, a rotação e combinações destas.

O **modelo projetivo** ou modelo homogêneo é um pouco mais elaborado em relação ao modelo Euclidiano, sobretudo por tratar subespaços planares que não necessariamente passam pela origem. Sua matriz de métrica é dada por

$$\begin{array}{c|ccccc} \cdot & e_0 & e_1 & e_2 & \dots & e_n \\ \hline e_0 & 1 & 0 & 0 & \dots & 0 \\ e_1 & 0 & 1 & 0 & \dots & 0 \\ e_2 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ e_n & 0 & 0 & 0 & \dots & 1 \end{array}$$

Na prática, o espaço de representação consiste em $d+1$ dimensões, enquanto o espaço base possui d dimensões. Essa dimensão extra, chamada coordenada homogênea (aqui representada por e_0) permite, além da interpretação de retas, planos e volumes (comumente chamados de *flats* na literatura), a definição de direções que, na prática, correspondem

a pontos cuja coordenada homogênea é zero, sendo interpretados no espaço base como pontos no infinito.

No que diz respeito às transformações suportadas, esse modelo também é mais poderoso que o modelo Euclidiano. Isso acontece porque a adição da coordenada homogênea permite a representação de subespaços planares como coordenadas afastadas da origem, o que imprime a ideia de translação, além da rotação em torno da origem, nativa do modelo Euclidiano. Ao conjunto de transformações implementadas nesse modelo é dado o nome de transformações de corpo rígido, mas apenas reflexões e rotações em planos que passam pela origem podem ser codificadas como versores.

O **modelo conforme** tem como matriz de métrica:

\cdot	n_o	e_1	e_2	\dots	e_d	n_∞
n_o	0	0	0	\dots	0	-1
e_1	0	1	0	\dots	0	0
e_2	0	0	1	\dots	0	0
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	0
e_d	0	0	0	\dots	1	0
n_∞	-1	0	0	0	0	0

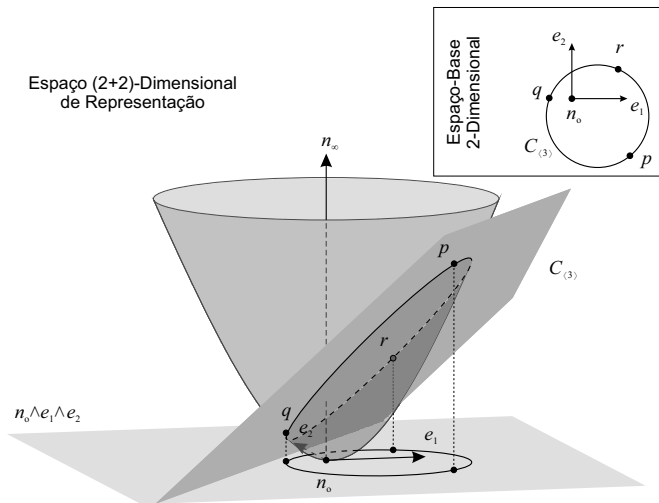


Figura 2.11: Representação de um círculo no modelo conforme 2-D, visto no espaço base como um produto externo entre três pontos.

Esse modelo é mais abrangente que os previamente citados, sendo capaz de representar por blades, além de *flats* e direções, circunferências reais e imaginárias (esses subespaços são comumente chamados de *rounds*) e subespaços tangentes. Seu espaço de representação consiste em $d + 2$ dimensões, sendo que as duas dimensões extras são representadas por vetores nulos.

Nesse espaço, as dimensões extras são vistas como extremos de um parabolóide, sendo um ponto na origem, denotado por n_o , e um no infinito, denotado por n_∞ . O espaço base é definido pela projeção das intersecções de primitivas com esse parabolóide no espaço base $e_1 \wedge e_2 \wedge \dots \wedge e_d$, como ilustrado na Figura 2.11. Perceba que um 3-blade criado pelo produto externo de três pontos, por exemplo, corresponde a um círculo quando projetado no espaço base. Essa observação permite concluir que é possível, num espaço com essas características, representar uma entidade geométrica não-linear (como um círculo) utilizando-se de primitivas puramente lineares (3-blades). O modelo conforme codifica na forma de versores, além de reflexão e rotação, também translação e escala uniforme, sendo que a reflexão pode ser em um plano ou em uma circunferência.

A Tabela 2.2 sumariza os modelos de geometria apresentados, os espaços vetoriais em que estão definidos, as interpretações geométricas provenientes de suas primitivas e as transformações que implementam na forma de versores.

2.4 Discussão

Ao longo deste Capítulo foram apresentados conceitos fundamentais à construção dos componentes da arquitetura de rede neural proposta.

O Capítulo possui uma fronteira implícita que o divide em dois momentos. A primeira parte explora características das camadas de redes neurais mais utilizadas, dentre elas as camadas de ativação, convolução, reamostragem e as camadas totalmente conectadas,

Tabela 2.2: Modelos de Geometria

Modelo	Espaço Vetorial	Primitivas	Transformações			
			Reflexão	Rotação	Translação	Escala Uniforme
Euclidiano	$\mathbb{R}^{n,0}$	Subespaços Planares	×	×		
Projetivo	$\mathbb{R}^{n+1,0}$	Subespaços Planares, Direções	×	×	×	
Conforme	$\mathbb{R}^{n+1,1}$	Subespaços Planares e Tangentes, Direções e Circunferências	×	×	×	×

bem como suas vantagens e desvantagens, significados práticos, e suas aplicações em redes neurais convolucionais cujas entradas são sinais discretos. A segunda parte tende a compor explicações sobre um conjunto de ferramentas matemáticas, como álgebra geométrica, empregados no desenvolvimento dos modelos propostos. A aplicação desses conceitos será discutida ao longo do Capítulo 4, que foca na aplicação dessas ferramentas nos modelos de componentes propostos.

Os elementos apresentados neste Capítulo são de fundamental importância para a compreensão da abordagem apresentada nesta tese. Ao passo em que a porção referente à álgebra geométrica permite a manipulação das transformações que modelam a ativação proposta como operações de tensores, a porção referente à convolução fornece um embasamento para as ConformalLayers enquanto composição de operações ao longo das camadas, o que é fundamental para a associatividade entre os conjuntos de tensores que representam as camadas da rede neural convolucional sequencial.

Capítulo 3

Trabalhos Relacionados

Nesse Capítulo serão apresentadas as abordagens recentes no contexto de compressão de redes neurais e redes neurais computacionalmente eficientes. Além disso, são explorados trabalhos que propõem modificações na arquitetura original de redes neurais convolucionais visando a obtenção de algum benefício específico. Por fim, são abordados os trabalhos que apresentam elementos de álgebra geométrica aplicados ao contexto de MLPs.

3.1 Compressão de Redes Neurais

Discussões e publicações recentes, sobretudo no contexto de computação de borda (do inglês, *edge computing*), motivaram a busca por modelos de redes neurais menores e/ou com consumo de energia reduzido durante os processos de treinamento e inferência, dadas as limitações desse tipo de computação.

A Hipótese do Bilhete de Loteria (do inglês, *Lottery Ticket Hypothesis*) [22] foi levantada em um dos artigos que motivou esse tipo de discussão. Segundo os autores, treinar uma rede neural com pesos criados aleatoriamente provavelmente trará bons resultados mas é provável que exista uma sub-rede que possa ser treinada com menos custo e com resultados similares. A analogia feita pelos autores é que treinar uma rede neural com pesos aleatórios é como comprar todos os bilhetes de loteria. É possível, no entanto, ser mais eficiente se houverem maneiras inteligentes de comprar os bilhetes. Nesse trabalho, os autores utilizam um esquema de poda da rede neural treinada para encontrar a sub-rede similar à rede completa. O processo é executado por força bruta, dado que as sub-redes são repetidamente treinadas e comparadas com a rede completa. Zhou *et al.* [23] apresentaram um estudo aprofundado nessa área, bem como outras abordagens para a seleção da sub-rede. Mehta [24] apresentou um critério de seleção das sub-redes

baseado em transferência de aprendizado de modo a evitar buscas exaustivas.

Em 2016, Hubara *et al.* [25] propuseram a redução do consumo de recursos e de energia por meio de um esquema de quantização. O processo é, em termos práticos, uma alteração no tipo de dados associados aos pesos da rede. Nesse trabalho, os pesos e as ativações são binarizados enquanto os coeficientes de erro da retro propagação do erro têm sua precisão reduzida.

Courbariaux *et al.* [26] propuseram um modelo baseado em redes cujos pesos e ativações são restritos a valores $+1$ ou -1 . Segundo os autores, esse processo reduz o consumo de memória, processamento e energia durante o processo de inferência, dado que passam a ser feitas operações binárias ao invés de produtos. Durante o processo de treinamento, no entanto, considerando que os pesos originais ainda devem ser mantidos como valores reais para fazer a retro propagação do erro, o custo é ligeiramente maior que o de uma rede comum.

Denton *et al.* [27] apresentaram um esquema de redução de tempo de inferência de redes neurais convolucionais. Os autores comprimem cada camada individualmente, através de decomposições em autovalores e autovetores, e ajustam as camadas seguintes de modo que a acurácia da rede seja restaurada. Eles reduziram pela metade o tempo de inferência ao custo de 1% da acurácia.

Omidvar *et al.* [28] basearam-se numa abordagem orientada ao particionamento de conjuntos de filtros convolucionais. Ao utilizarem essas partições de conjuntos como entradas de uma rede neural auxiliar, geraram filtros convolucionais reutilizáveis. Esse processo diminui consideravelmente a quantidade de parâmetros treináveis das redes neurais convolucionais. Tetko [29] apresentou uma técnica baseada na agregação de dois métodos de aprendizado, o chamado *ensemble*. Essa técnica em particular agrega redes neurais e busca pelos k vizinhos mais próximos e utiliza a distância entre os dados inferidos e os dados rotulados para ajustar o viés das redes neurais.

3.2 Redes Neurais Computacionalmente Eficientes

Em 2014, Springenberg *et al.* [30] mostraram que é possível construir uma rede neural apenas com convoluções e MLP, eliminando a necessidade de camadas de reamostragem (*pooling*) e regularização, por exemplo. Além disso, demonstraram também que uma MLP pequena conectada à porção convolucional já atinge bons resultados quando comparados com o estado-da-arte. Nesse processo, no entanto, ainda se faz necessária a utilização de

funções de ativação não-lineares.

Mathieu *et al.* [31] apresentaram uma arquitetura para redes neurais convolucionais baseada na representação das convoluções no domínio frequência. A grande vantagem desse esquema é a possibilidade de aplicar as convoluções que, no domínio frequência, são representadas como o produto de Hadamard entre duas matrizes, em conjuntos de imagens simultaneamente (execução em *mini-batch*). Além disso, a existência de ferramentas para transformadas rápidas de Fourier em GPU favorece a aplicação dessa arquitetura. Uma das desvantagens desse modelo é a necessidade de manter os filtros no domínio frequência que, por terem que ter o mesmo tamanho das entradas, acabam por consumir mais memória.

YOLO [32] é um modelo de detecção e com inferência em tempo real. Na prática, o modelo detecta a região em que está contido o objeto de interesse e a classificação simultaneamente e utilizando a mesma rede neural. A arquitetura dessa rede consiste em uma sequência de redes neurais com algumas das suas conexões ignoradas, de maneira similar a uma rede neural residual [33]. Wu *et al.* apresentaram a SqueezeDet [34], uma abordagem que melhora a detecção de objetos e é mais voltada ao contexto de veículos autônomos. De acordo com os autores, apesar da similaridade com a YOLO, sua técnica é capaz de encontrar mais regiões de interesse utilizando menos parâmetros.

Saha *et al.* [35] apresentaram um problema relacionado à inferência de modelos em dispositivos de borda. Esses dispositivos, que podem ser veículos autônomos, *smartwatches*, roteadores e celulares, por exemplo, possuem baterias com carga limitada e pouca memória RAM disponível. Assim, eles propuseram uma camada de rede neural de reamostragem baseada em redes neurais recorrentes para fazer uma subamostragem com custo computacional reduzido mantendo quase a mesma acurácia quando comparadas com camadas de reamostragem tradicionais.

Alwani *et al.* [36] apresentaram uma reformulação do modelo de treinamento de redes neurais convolucionais nos chamados hardware aceleradores (GPUs, FPGAs etc.). Baseando-se na ideia de que cada ponto de uma camada oculta depende de regiões específicas da entrada, os autores criaram uma região em formato de pirâmide que passa pelas várias camadas. Assim, ao alterar o modelo de cópia dos dados de entrada da memória principal para a memória do hardware acelerador, foi possível tirar proveito dos esquemas de cache já implementados a nível de hardware.

Tan and Le [37] propuseram a EfficientNet, uma abordagem para escalar redes neurais utilizando uma agregação de parâmetros. Ao reescalar o tamanho de redes neurais utili-

zando uma quantidade fixa de recursos, foi possível manter a acurácia do estado-da-arte utilizando uma quantidade consideravelmente menor de recursos (cerca de uma ordem de grandeza).

MobileNets [38] são redes neurais convolucionais desenhadas especificamente para dispositivos com recursos computacionais limitados. Nessa abordagem, são utilizados dois hiperparâmetros relacionados diretamente à profundidade do modelo e à resolução das imagens de entrada, sendo possível obter modelos que se ajustem às restrições de aplicação e de hardware ao passo que mantém o tempo de inferência e a acurácia dentro dos limites aceitáveis, dependendo da aplicação.

3.3 Aplicações de Álgebra Geométrica em Redes Neurais

Person *et al.* [39] apresentaram um modelo de retro propagação escrito em álgebra geométrica. O modelo é puramente matemático, ignorando características biológicas. Os autores argumentam, no entanto, que essa modelagem possibilita a representação da informação em dimensionalidades mais altas, suportadas pela álgebra geométrica.

Buchholz [40, 41] formulou um esquema de MLP em sua tese condicionado ao fato de que cada neurônio, por si só, já modela uma transformação geométrica. Assim, apresentou um estudo aprofundado sobre *Spinor Clifford Neurons* (SCN), em que os pesos associados a cada neurônio atuam como rotores. Além disso, o autor utiliza funções hiperbólicas como função de ativação. Essa arquitetura estende para o caso n -dimensional redes onde os coeficientes são valores reais (caso 0-dimensional), números complexos (caso 2-dimensional) ou quatérnios (caso 3-dimensional).

Rubio *et al.* [42] publicaram em 2012 um modelo de neurônio, e sua agregação, como um MLP, baseado no modelo de cônicas da álgebra geométrica. Os autores utilizam um esquema não-linear de distância entre pontos e uma região de decisão gerada a partir de uma seção cônica como conceito de função de ativação. Segundo os autores, o algoritmo apresenta uma capacidade de generalização melhor que um MLP comum mas ainda demonstra restrições quanto a dados não linearmente separáveis.

3.4 Discussão

Nesse Capítulo, foram apresentadas algumas das abordagens para redes neurais que estão relacionados aos problemas atacados nessa tese.

Os trabalhos abordados ao longo da Seção 3.1 exploram o problema da compressão de redes neurais. Essas abordagens são baseadas, sobretudo, em redes já treinadas e buscam reduzir o tempo e o custo computacional de processos de inferência com base em seleções de sub-redes ou quantizações de dados. A maior parte dessas técnicas, no entanto, se mostra mais custosa em termos de memória uma vez que ainda necessita que a rede original seja mantida durante o processo de treinamento. Outras técnicas se baseiam em *ensemble* de algoritmos, o que acaba por agravar o problema do custo computacional. É importante enfatizar que os trabalhos de Omivar *et al.* [28] e Tetko [29], apesar de trazerem consigo o termo "associatividade", não fazem uso da propriedade associativa, descrita na Equação 1.1. A Seção 3.2 aborda técnicas para construção de arquiteturas que visam reduzir o custo computacional associadas ao treinamento e à inferência nessas redes. Além disso, aborda também arquiteturas otimizadas, sejam para dispositivos móveis, sejam para hardwares aceleradores. Esse tipo de técnica, apesar de sua eficiência, dificulta a portabilidade de modelos pré-treinados entre os diversos dispositivos.

A Seção 3.2 enumera alguns dos avanços mais recentes no contexto de redes neurais convolucionais, principalmente no que diz respeito à arquitetura, como a substituição de componentes não-lineares, como a re-amostragem por máximos (*max-pooling*) pela convolução, que é uma operação linear. Além disso, trabalhos recentes [43] também mostraram que é possível substituir a convolução propriamente dita por sua versão no domínio frequência, trazendo algumas vantagens ao processo de inferência, ao custo de memória. Os trabalhos mais recentes nessa área também propõem modificações no esquema de treinamento e inferência de redes neurais visando uma redução do tamanho do modelo e do tempo de inferência ao passo em que a acurácia se mantém próxima à original. Esse tipo de objetivo reforça a relevância do trabalho apresentado aqui, posto que é um problema recente e que permanece em aberto.

Por fim, a Seção 3.3 apresenta os trabalhos mais recentes envolvendo álgebra geométrica no contexto de redes neurais, evidenciando as vantagens da aplicação dessa álgebra, sobretudo quanto ao suporte às dimensionalidades mais altas e à possibilidade de prover uma interpretação geométrica às operações executadas nas camadas. Ao contrário dessas abordagens, que aplicam álgebra geométrica como formalismo matemático

para generalização, mas mantendo a não-linearidade de alguma etapas, neste trabalho é considerada inicialmente a aplicação de modelos de geometria codificados em álgebra geométrica tanto para generalização quanto para linearização dos processos para, posteriormente, re-codificar o modelo como álgebra de tensores, visando a simplificação do processo como um todo e possibilitando sua implementação com as bibliotecas de funções disponíveis atualmente.

Capítulo 4

ConformalLayers

Esse Capítulo apresenta, em detalhes, a abordagem principal da tese. A construção geométrica e matemática da função de ativação não-linear e diferenciável, a ReSPro, é apresentada ao longo da Seção 4.1. Essa construção se baseia na premissa de que os dados de entrada da rede são valores discretos e geometricamente interpretados como pontos em \mathbb{R}^n . A Seção 4.2, por sua vez, explora a composição tensorial da ReSPro, ao passo em que a Seção 4.3 reapresenta as operações em camadas lineares enquanto tensores. A Seção 4.4 explora a modelagem da composição das ConformalLayers como um todo, partindo da ReSPro e agregando as demais operações como convolução e reamostragem, por exemplo. Por fim, a Seção 4.5 apresenta alguns detalhes da implementação das ConformalLayers.

4.1 ReSPro: de Transformações a Funções de Ativação

A função ReSPro (acrônimo para *Reflection*, *Scaling* e *Projection*, que corresponde às operações de reflexão, escala e projeção, cuja composição define a modelagem da função) é apresentada nesta Seção. Essa função corresponde à uma agregação de duas transformações definidas em dimensionalidades maiores que a do espaço base e de maneira bastante específica, produzindo uma modelagem linear com interpretação geométrica não-linear, desejada em funções de ativação.

A Figura 4.1 apresenta o mapeamento passo a passo feito pela ReSPro. Visando manter a simplicidade do exemplo, foi assumida dimensionalidade $d = 1$ para o espaço cartesiano onde o dado de entrada reside, chamado aqui de espaço base.

Considere agora dois pontos, z e x , sobre o eixo e_d (Figura 4.1a) e que distam, respectivamente, α e δ unidades da origem, sendo $\delta \in [0, \alpha]$. Considere agora a adição de uma

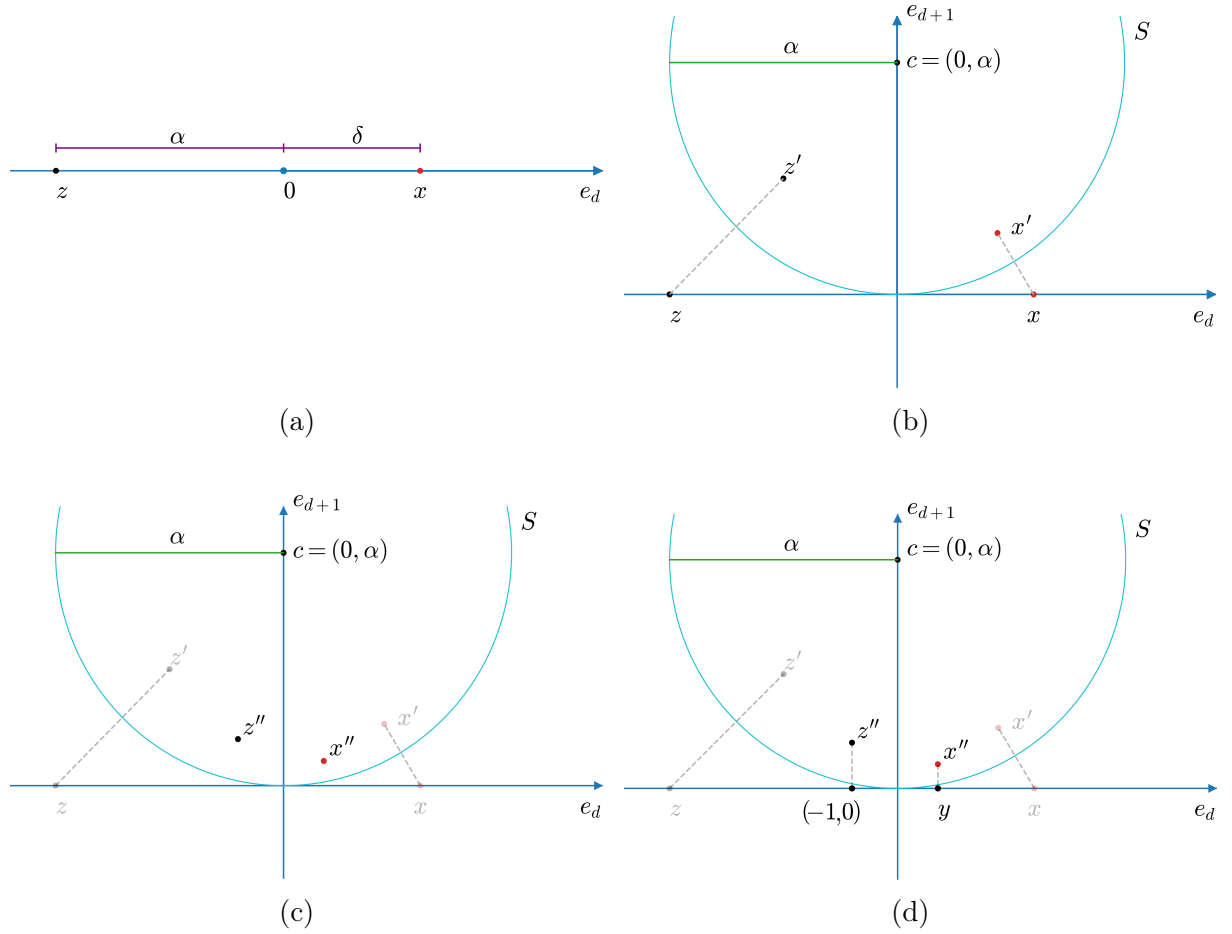


Figura 4.1: Passo a passo da intuição para a formulação da ReSPro

dimensão extra e_{d+1} ao espaço cartesiano seguida pela inclusão de uma hipersfera S (um círculo, nesse exemplo) com raio α e centrada em $c = (0, \alpha)$. Levando-se em conta essa dimensão extra, os pontos z e x são agora representados como $z = (z_d, z_{d+1}) = (-\alpha, 0)$ e $x = (x_d, x_{d+1}) = (\delta, 0)$.

A Figura 4.1b ilustra como a utilização da hipersesfera S como um espelho para refletir os pontos z e x , que originalmente estavam fora da hipersesfera, leva aos seus reflexos z' e x' , respectivamente, no interior de S . É possível observar um comportamento interessante nessa transformação: se um ponto está próximo à hipersesfera, seu reflexo também estará; por outro lado, pontos afastados da hipersesfera terão seus reflexos mais próximos à origem. A relação entre a distância entre os pontos e a hipersesfera e a distância de seus reflexos do centro da hipersesfera é não-linear. No limite, pontos no infinito são refletidos para o centro c ao passo em que pontos sobre a hipersesfera S permanecem refletidos sobre a hipersesfera. Pela construção específica desse espaço, não é necessário se preocupar com a reflexão de pontos dentro da hipersesfera, posto que S é tangente ao espaço expandido por $\{e_1, e_2, \dots, e_d\}$, e todas as entradas residem nesses espaço (*i.e.*, a coordenada e_{d+1} desses

pontos é sempre zero).

Uma vez obtidos os pontos refletidos na hipersfera, duas transformações lineares são aplicadas sobre esses pontos. A primeira delas é uma transformação de escala uniforme por um fator de $2/\alpha$. Essa transformação está ilustrada na Figura 4.1c, que mapeia os pontos z' e x' para z'' e x'' , respectivamente. A Figura 4.1d apresenta a última transformação da ReSPro: uma projeção ortogonal dos pontos obtidos no passo anterior no espaço cartesiano d -dimensional original. Perceba que z'' é projetado em $(-1, 0)$ enquanto x'' é projetado em $y = (y_d, 0)$, *i.e.*, a projeção demonstrada na Figura 4.1d torna as coordenadas associadas à dimensão extra e_{d+1} iguais a zero de modo que, por serem constantes, podem ser removidas da representação final. Assim, formalmente, a ReSPro é definida como um mapeamento

$$f : x \rightarrow y, \text{ sujeito a } \|x\|_2 \in [0, \alpha] \text{ e } \|y\|_2 \in [0, 1], \quad (4.1)$$

onde $x, y \in \mathbb{R}^d$ são, respectivamente, os pontos que representam os dados de entrada e saída da função e $\|p\|_2$ representa a norma L^2 de p .

A função ReSPro é definida sobre a região do espaço cartesiano d -dimensional onde $\|x\|_2 \in [0, \alpha]$. Assim, as coordenadas $x_i \in [-\alpha, +\alpha]$ com $i \in \{1, 2, \dots, d\}$ possuem, como restrição, o fato da distância euclidiana do ponto x até a origem do espaço não ser maior do que α unidades. As coordenadas resultantes são $y_i \in [-1, +1]$ tal que $\|y\|_2 \in [0, 1]$.

ReSPro é uma função (i) diferenciável, (ii) representável por tensores de maneira associativa, (iii) não-linear e (iv) inversível. A verificação da propriedade (i) é trivial, bastando para isso considerar que cada transformação que ocorre na ReSPro é diferenciável [21, 44], de modo que a transformação como um todo também é. As propriedades (ii), (iii) e (iv) são demonstradas na Seção 4.2.

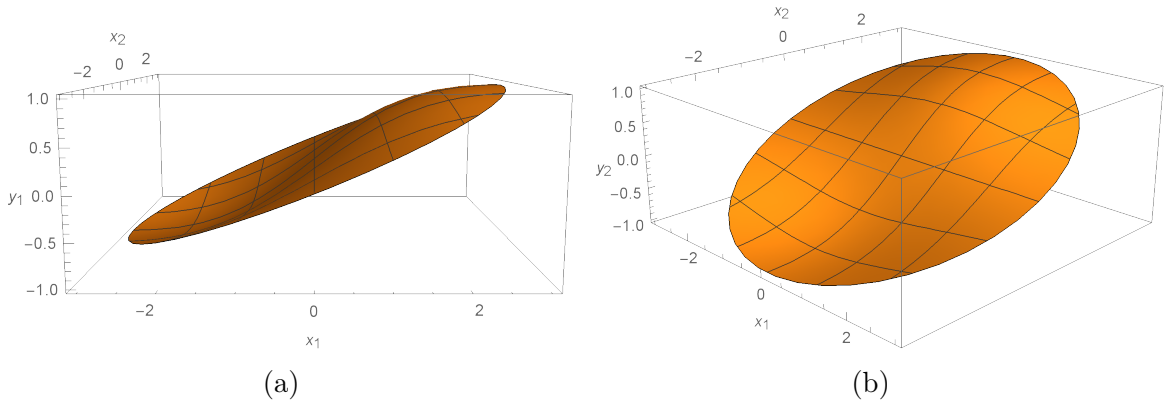


Figura 4.2: Representação gráfica do mapeamento produzido pela ReSPro. As superfícies (a) e (b) mostram, respectivamente, os valores dos coeficientes y_1 e y_2 do ponto y resultante da aplicação da ReSPro sobre o ponto $x = (x_1, x_2)$, assumido-se $\alpha = 3$, *i.e.*, $\|x\|_2 \leq 3$.

Outra característica da ReSPro vem do fato de que, por construção, trata-se de uma função de ativação global. Em termos práticos, diferentemente de outras abordagens que aplicam a função de ativação por elemento do vetor de entrada, a função ReSPro ativa todos os coeficientes do vetor de entrada simultaneamente. Isso é equivalente a ter uma ativação por elemento seguida por uma normalização para garantir $\|y\|_2 \in [0, 1]$.

A Figura 4.2 representa graficamente a saída da ReSPro para o ponto $x = (x_1, x_2)$ em \mathbb{R}^2 com $\alpha = 3$. O formato de disco do gráfico é proveniente do domínio em que a função está definida, *i.e.*, $\|x\|_2 \in [0, \alpha]$.

4.2 ReSPro: de Funções de Ativação a Tensores

A Seção 4.1 apresentou a geometria e a ideia geral por trás da ReSPro. Esta Seção, por sua vez, visa detalhar a álgebra por trás das transformações que compõem a ReSPro. Assim, pode-se dizer que, ao passo em que a primeira Seção traz a intuição sobre a função por meio de exemplos, esta Seção descreve a obtenção da ReSPro com o formalismo matemático necessário para a ampla compreensão desta tese e as demonstrações de suas propriedades. Para isso, é feito uso dos conceitos apresentados no Capítulo 2.

Considere, novamente, um ponto que representa a entrada da ReSPro. Esse ponto é representado pelo vetor $x = (x_1, x_2, \dots, x_d)$, definido em \mathbb{R}^d . O passo seguinte é adicionar uma dimensão extra em \mathbb{R}^d , levando a um espaço \mathbb{R}^{d+1} . Considere agora o mergulho desse ponto num modelo conforme. Como discutido na Seção 2.3.6, esse novo espaço possui mais duas coordenadas extra, uma associada ao ponto na origem, n_o , e uma associada a um ponto no infinito, n_∞ . Assim, o espaço de representação resultante possui os vetores de base $\{e_1, e_2, \dots, e_d, e_{d+1}, n_o, n_\infty\}$ e matriz de métrica:

\cdot	e_1	e_2	\dots	e_{d+1}	n_o	n_∞
e_1	1	0	\dots	0	0	0
e_2	0	1	\dots	0	0	0
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots
e_{d+1}	0	0	\dots	1	0	0
n_o	0	0	\dots	0	0	-1
n_∞	0	0	\dots	0	-1	0

onde \cdot denota o produto interno de vetores. A representação do ponto finito x mergulhado

nesse espaço é dada pelo vetor

$$V = x'_1 e_1 + x'_2 e_2 + \cdots + x'_d e_d + x'_o n_o - \frac{x'_o}{2} \sum_{i=1}^d (x_i)^2 n_\infty, \quad (4.2)$$

onde $x_i = x'_i/x'_o$, para $i \in \{1, 2, \dots, d+1\}$, e $x'_o \neq 0$. Por definição, é assumido que $x_{d+1} = 0$. A álgebra geométrica construída sobre o espaço conforme permite que transformações como reflexões, rotações e translações sejam representadas como versores [21] (ver Tabela 2.2). Assim, ao longo desta Seção, as expressões matemáticas serão escritas utilizando-se álgebra geométrica e, posteriormente, mapeadas para operações com tensores. Os termos nas expressões a seguir representam blades ou versores codificados como multivetores no espaço multivetorial $\bigwedge \mathbb{R}^{d+2,1}$, considerando a dimensão extra e_{d+1} e as duas associadas ao modelo conforme, n_o e n_∞ . A notação das operações pode ser revisitada na Seção 2.3.

Seja V' um 2-blade que codifica um ponto planar obtido de V e n_∞ utilizando o produto externo (ver Seção 2.3.2):

$$\begin{aligned} V' &= V \wedge n_\infty \\ &= (x'_1 e_1 + x'_2 e_2 + \cdots + x'_d e_d + x'_o n_o + x'_\infty n_\infty) \wedge n_\infty \\ &= (x'_1 e_1 + x'_2 e_2 + \cdots + x'_d e_d + x'_o n_o) \wedge n_\infty \\ &= v \wedge n_\infty + x'_o (n_o \wedge n_\infty) \\ &= (v + x'_o n_o) \wedge n_\infty, \end{aligned} \quad (4.3)$$

onde $v = x'_1 e_1 + x'_2 e_2 + \cdots + x'_d e_d$, codificando toda a porção euclidiana d -dimensional do espaço, e $x'_\infty = -\frac{x'_o}{2} \sum_{i=1}^d (x_i)^2$. Perceba que V' , na prática, codifica um par de pontos em que um deles está no infinito.

Considere agora uma hiperesfera centrada em $c = (0, 0, \dots, \alpha) \in \mathbb{R}^{d+1}$ e raio α , para $\alpha \geq 0$. A Figura 4.1b mostra o caso $(1+1)$ -dimensional. A operação de reflexão nessa hiperesfera é representada pelo versor T_{Re} , dado por [21, Capítulo 16]

$$T_{Re} = \alpha e_{d+1} + n_o. \quad (4.4)$$

A operação de escala uniforme por um fator $2/\alpha$, por sua vez, é representada pelo versor T_S , dado por [21, Capítulo 16]

$$T_S = \cosh\left(\frac{1}{2} \log \frac{2}{\alpha}\right) + \sinh\left(\frac{1}{2} \log \frac{2}{\alpha}\right) (n_o \wedge n_\infty) \quad (4.5)$$

Assim, a construção de uma operação de reflexão na hiperesfera seguida da escala uniforme

é dada por

$$\begin{aligned} T &= T_S T_{Re} \\ &= \left(\cosh \left(\frac{1}{2} \log \frac{2}{\alpha} \right) + \sinh \left(\frac{1}{2} \log \frac{2}{\alpha} \right) (n_o \wedge n_\infty) \right) (\alpha e_{d+1} + n_o), \end{aligned} \quad (4.6)$$

cujo inverso é dado por

$$\begin{aligned} T^{-1} &= (T_S T_{Re})^{-1} \\ &= \left(\frac{1}{\alpha} e_{d+1} + \frac{1}{\alpha^2} n_o \right) \left(\cosh \left(\frac{1}{2} \log \frac{2}{\alpha} \right) - \sinh \left(\frac{1}{2} \log \frac{2}{\alpha} \right) (n_o \wedge n_\infty) \right)^{-1}. \end{aligned} \quad (4.7)$$

A Equação 2.32 mostra a aplicação de transformações representadas como versores em álgebra geométrica. Aplicando-se essa equação a esse contexto, temos

$$\begin{aligned} V'' &= T V' T^{-1} \\ &= T (V \wedge n_\infty) T^{-1} \\ &= (-T V T^{-1}) \wedge (-T n_\infty T^{-1}) \\ &= \frac{2}{\alpha} (v \wedge e_{d+1}) + \frac{1}{\alpha} (v \wedge n_o) + \frac{2}{\alpha} (v \wedge n_\infty) - x'_o (e_{d+1} \wedge n_o) + x'_o (n_o \wedge n_\infty). \end{aligned} \quad (4.8)$$

É válido observar que V'' , da mesma forma que V' , corresponde a um par de pontos. Como mencionado na Seção 4.1, pontos no infinito, quando refletidos numa hiperesfera, são mapeados para o centro dessa hiperesfera. Assim, em V'' , um comportamento similar é observado em um dos pontos do par, com a ressalva de que o par de pontos refletidos foram também transformados por uma escala uniforme. Esse é o ponto conhecido do par. O objetivo, assim, é determinar o outro ponto do par. Esse processo se dá pela extração do bissetor perpendicular ao par de pontos. Ao contrair n_∞ de V'' (ver contração à esquerda, na Seção 2.3.3) é obtido o dual (ver dualização, na Seção 2.3.4) do bissetor perpendicular ao par de pontos V'' :

$$\begin{aligned} H &= n_\infty \rfloor V'' \\ &= n_\infty \rfloor \left(\frac{2}{\alpha} (v \wedge e_{d+1}) + \frac{1}{\alpha} (v \wedge n_o) + \frac{2}{\alpha} (v \wedge n_\infty) - x'_o (e_{d+1} \wedge n_o) + x'_o (n_o \wedge n_\infty) \right) \\ &= \frac{1}{\alpha} v - x'_o e_{d+1} - x'_o n_\infty. \end{aligned} \quad (4.9)$$

Esse bissetor pode ser usado como um “espelho” para refletir o ponto conhecido do par V' (o centro da hiperesfera multiplicado por um fator de escala) para obter o ponto até então desconhecido. Para isso, primeiro utiliza-se o produto de versores para se determinar para

onde a transformação T levou o ponto que estava originalmente no infinito. Essa operação pode ser descrita como $-Tn_\infty T^{-1}$. A utilização de uma entidade geométrica nesse espaço atuando como espelho para uma reflexão é definida na Equação 2.32 e é dada por

$$p = -H \left(-Tn_\infty T^{-1} \right) H. \quad (4.10)$$

Em seguida, um produto externo com n_∞ simplifica a expressão ao converter o ponto finito em um ponto *flat*. Por fim, é feita uma projeção ortográfica no espaço original $e_1 \wedge e_2 \wedge \dots \wedge e_d \wedge n_o$. Essa composição de operações pode ser descrita como

$$\begin{aligned} V''' &= \text{PROJECT} (p \wedge n_\infty) \\ &= \text{PROJECT} \left(\left(-H \left(-Tn_\infty T^{-1} \right) H \right) \wedge n_\infty \right) \\ &= \text{PROJECT} \left(\left(-H \left(\frac{2}{\alpha} e_{d+1} + \frac{1}{\alpha} n_o + \frac{2}{\alpha} n_\infty \right) H \right) \wedge n_\infty \right) \\ &= \text{PROJECT} \left(\left(-\frac{2x'_o}{\alpha^2} v - \frac{2(v \cdot v)}{\alpha^3} e_{d+1} - \frac{1}{\alpha} \left(\frac{(v \cdot v) x'_o}{\alpha^2} + x_o'^2 \right) n_o - \frac{2(v \cdot v)}{\alpha^3} n_\infty \right) \wedge n_\infty \right) \\ &= \text{PROJECT} \left(-\frac{2x'_o}{\alpha^2} (v \wedge n_\infty) - \frac{2(v \cdot v)}{\alpha^3} (e_{d+1} \wedge n_\infty) - \left(\frac{(v \cdot v) x'_o}{\alpha^3} + \frac{x_o'^2}{\alpha} \right) (n_o \wedge n_\infty) \right) \\ &= -\frac{2x'_o}{\alpha^2} (v \wedge n_\infty) - \left(\frac{(v \cdot v) x'_o}{\alpha^3} + \frac{x_o'^2}{\alpha} \right) (n_o \wedge n_\infty). \end{aligned} \quad (4.11)$$

Uma observação importante vem do fato de que, ao multiplicar o ponto *flat* V''' por um valor escalar diferente de zero, ele permanece um ponto *flat*. É possível tirar proveito disso para simplificar a expressão usando $-\alpha^2/(2x'_o)$ como fator de escala, levando ao ponto *flat* Y , dado por

$$\begin{aligned} Y &= -\frac{\alpha^2}{2x'_o} V''' \\ &= -\frac{\alpha^2}{2x'_o} \left(-\frac{2x'_o}{\alpha^2} (v \wedge n_\infty) - \left(\frac{(v \cdot v) x'_o}{\alpha^3} + \frac{x_o'^2}{\alpha} \right) (n_o \wedge n_\infty) \right) \\ &= (v \wedge n_\infty) + \left(\frac{(v \cdot v)}{2\alpha} + \frac{\alpha x_o'}{2} \right) (n_o \wedge n_\infty) \\ &= \left(v + \left(\frac{(v \cdot v)}{2\alpha} + \frac{\alpha x_o'}{2} \right) n_o \right) \wedge n_\infty. \end{aligned} \quad (4.12)$$

Por fim, o ponto *flat* Y , na Equação (4.12), pode ser reescrito utilizando tensores para

representar a aplicação da função ReSPro sobre X :

$$Y = \begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_d \\ y'_o \end{pmatrix} = \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_d \\ \frac{\alpha}{2}x'_o + \frac{1}{2\alpha} \sum_{i=1}^d (x'_i)^2 \end{pmatrix} \quad (4.13)$$

$$Y = \left(\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & \frac{\alpha}{2} \end{pmatrix} + \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ \frac{x'_1}{2\alpha} & \frac{x'_2}{2\alpha} & \cdots & \frac{x'_d}{2\alpha} & 0 \end{pmatrix} \right) \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_d \\ x'_o \end{pmatrix} \quad (4.14)$$

$$= (F_M + F_T X) X, \quad (4.15)$$

sendo F_M um tensor esparsos de ordem 2, *i.e.*, uma matriz e F_T um tensor esparsos de ordem 3 quase todo preenchido com zeros, exceto por sua última fatia, dada por

$$F_{T[d+1]} = \begin{pmatrix} \frac{1}{2\alpha} & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{2\alpha} & 0 \\ 0 & \cdots & 0 & 0 \end{pmatrix}.$$

Note que, nessa representação utilizando tensores, é necessária apenas uma dimensão extra em relação ao espaço base original d -dimensional, *i.e.*, o espaço utilizado nessa representação é o \mathbb{R}^{d+1} , diferentemente do espaço do modelo de geometria conforme, $\mathbb{R}^{d+2,1}$, que seria necessário considerando as operações demonstradas. Assim, a utilização da construção com tensores possibilita que a aplicação da ReSPro seja feita diretamente com os tensores, sem a necessidade de passar explicitamente por todas as operações de álgebra geométrica que a compõem, descritas ao longo desta Seção. Em termos práticos, isso significa que a utilização da ReSPro é totalmente linear, com base na operação dos tensores F_M e F_T , ao passo em que a interpretação é não-linear. A demonstração da linearidade e da inversibilidade da ReSPro são apresentadas a seguir.

Não-linearidade: Para uma função ser linear, é necessário que sejam satisfeitas duas condições [45]:

1. $f(cx) = cf(x) \forall c, x \in \mathbb{R}$; e
2. $f(x + y) = f(x) + f(y) \forall x, y \in \mathbb{R}$.

Sem perda de generalidade, seja $X = (x, 1)^\top$ um vetor $(1 + 1)$ -dimensional representando uma entrada para a ReSPro. O vetor Y de saída da ReSPro utilizando X como entrada é dado por:

$$Y = \begin{pmatrix} x \\ \frac{\alpha}{2}x'_o + \frac{1}{2\alpha} \sum_{i=1}^1 (x'_i)^2 \end{pmatrix} = \begin{pmatrix} x \\ \frac{\alpha}{2} + \frac{1}{2\alpha}x^2 \end{pmatrix}. \quad (4.16)$$

O valor y de saída da ReSPro será dado, assim, por:

$$y = f(x) = \frac{x}{\frac{\alpha}{2} + \frac{1}{2\alpha}x^2} = \frac{2x\alpha}{x^2 + \alpha^2}. \quad (4.17)$$

Ao verificar-se a primeira condição de linearidade, verifica-se que

$$f(cx) = \frac{2cx\alpha}{c^2x^2 + \alpha^2} \quad (4.18)$$

ao passo em que

$$cf(x) = c \frac{2x\alpha}{x^2 + \alpha^2} = \frac{2cx\alpha}{x^2 + \alpha^2}. \quad (4.19)$$

Assim, posto que $f(cx) \neq cf(x)$, *i.e.*, ocorre uma violação da primeira condição de linearidade, demonstra-se que a **ReSPro é uma função não-linear**.

Inversibilidade: Para uma função ser inversível, é necessário que a seguinte condição seja satisfeita [46]

$$f(a) = f(b) \iff a = b. \quad (4.20)$$

Para essa demonstração, é possível utilizar a construção apresentada na Equação 4.18:

$$f(a) = \frac{2a\alpha}{a^2 + \alpha^2}, \text{ e} \quad (4.21)$$

$$f(b) = \frac{2b\alpha}{b^2 + \alpha^2} \quad (4.22)$$

Expandindo-se essas representações e manipulando algebricamente, temos

$$\begin{aligned} f(a) &= f(b) \\ \frac{2a\alpha}{a^2 + \alpha^2} &= \frac{2b\alpha}{b^2 + \alpha^2} \\ \frac{a}{a^2 + \alpha^2} &= \frac{b}{b^2 + \alpha^2} \end{aligned}$$

$$\begin{aligned}
\frac{a}{a^2 + \alpha^2} - \frac{b}{b^2 + \alpha^2} &= 0 \\
a(b^2 + \alpha^2) - b(a^2 + \alpha^2) &= 0 \\
ab^2 + a\alpha^2 - a^2b - b\alpha^2 &= 0 \\
(a - b)(\alpha^2 - ab) &= 0.
\end{aligned} \tag{4.23}$$

A Equação 4.23 só pode ser verdadeira em duas situações:

$$(a - b) = 0 \rightarrow a = b, \text{ ou} \tag{4.24}$$

$$(\alpha^2 - ab) = 0 \rightarrow \alpha^2 = ab \tag{4.25}$$

O primeiro caso apresenta, de maneira trivial, a possibilidade que satisfaz a condição apresentada na Equação 4.20. O segundo caso demanda uma análise um pouco mais aprofundada. Sabe-se que

$$\alpha \geq a \rightarrow \alpha^2 \geq a^2, \text{ e que} \tag{4.26}$$

$$\alpha \geq b \rightarrow \alpha^2 \geq b^2. \tag{4.27}$$

Substituindo-se a Equação 4.25 nas Equações 4.26 e 4.27, temos

$$ab \geq a^2 \rightarrow b \geq a, \text{ e} \tag{4.28}$$

$$ab \geq b^2 \rightarrow a \geq b. \tag{4.29}$$

A única solução possível que resolva ambas as inequações simultaneamente ocorre se $a = b$. Desse modo, observando-se que a condição para inversibilidade apresentada na Equação 4.20 é satisfeita, demonstra-se que a **ReSPro é uma função inversível**.

Apesar das demonstrações de não-linearidade e inversibilidade terem sido apresentadas para o caso $(1 + 1)$ -dimensional e com coordenada homogênea igual a 1, é possível verificar facilmente que essas propriedades se mantêm, mesmo fora desses casos.

4.3 De Camadas Lineares a ConformalLayers em Redes Neurais Convolucionais

Uma vez apresentada a ReSPro como uma composição de tensores (Equação 4.15), a etapa seguinte consiste em representar as demais operações lineares comuns em redes neurais convolucionais também como tensores. Esse é um segundo passo na direção de uma arquitetura de composição de camadas totalmente linear e associativa, as ConformalLayers.

Assim, ao longo dessa Seção, é explorada essa representação tensorial de cada componente de modo que, nas próximas seções, será apresentada a composição final agregando todas as camadas.

A representação das operações comumente utilizadas em redes neurais como matrizes já é uma abordagem conhecida. Matrizes de Toeplitz, por exemplo, codificam convoluções discretas num espaço n -dimensional e podem ser modificadas para codificarem a correlação cruzada válida [2]. A reamostragem por média (*average pooling*), por sua vez, pode ser modelada como um filtro de média [47] como um caso particular de convolução com pesos constantes e passo específico. Configurações como preenchimento (*padding*), dilatação (*dilation*) e passo (*stride*) podem ser codificadas como matrizes compostas por zeros e uns. Assim, sem perda de generalidade, tanto as entradas quanto as saídas dessas modelagens são representadas como sinais discretos de apenas uma dimensão e um canal (vetores). A seguir são apresentadas as modelagens em 1-D da convolução, reamostragem por média e regularização por *dropout*, bem como os tensores que modelam alguns dos parâmetros dessas operações. A modelagem de operações com dimensionalidades maiores e mais canais pode ser encontrada em [48].

- **Pesos** Os pesos são modelados como vetores em \mathbb{R}^{d_w+1} , sendo d_w o tamanho do filtro. O vetor que modela o conjunto de pesos é dado por

$$W = (w_1, w_2, \dots, w_{d_w}, 1)^\top, \quad (4.30)$$

onde \square^\top corresponde à transposição.

- **Preenchimento com zeros (*padding with zeros*)** A matriz $P = (p_{ij})$ é uma matriz constante de tamanho $(d_x + 2\delta_P + 1) \times (d_x + 1)$ que modela o preenchimento de δ_P unidades com zeros, onde δ_P é o número de zeros adicionados ao final de cada sinal representado como um vetor em \mathbb{R}^{d_x+1} . P_r e P_c representam, respectivamente, o número de linhas e colunas de P . Os elementos da matriz que modela o preenchimento com zeros do sinal são dados por

$$p_{ij} = \begin{cases} 1 & , \text{ para } ((i - \delta_P) = j \text{ e } i < P_r \text{ e } j < P_c) \text{ ou } (i = P_r \text{ e } j = P_c), \\ 0 & , \text{ caso contrário.} \end{cases} \quad (4.31)$$

- **Dilatação (*dilation*)** A matriz $D = (d_{ij})$, de tamanho $(d_w + 1) \times ((d_w - 1)\delta_D + 2)$, modela a dilatação com taxa δ_D . D_r e D_c representam, respectivamente, o número de linhas e colunas de D . Os elementos da matriz que modela a dilatação de um

sinal são dados por

$$d_{ij} = \begin{cases} 1 & , \text{ para } ((i-1)\delta_D + 1 = j \text{ e } i < D_r \text{ e } j < D_c) \text{ ou } (i = D_r \text{ e } j = D_c), \\ 0 & , \text{ caso contrário.} \end{cases} \quad (4.32)$$

- **Passo (*stride*)** A matriz $S = (s_{ij})$ é uma matriz constante de tamanho $(\lfloor \frac{(d_x + 2\delta_P - \delta_D(d_w - 1) - 1)}{\delta_S} \rfloor + 2) \times (d_x - (d_w - 1)\delta_D + 2\delta_P + 1)$ que modela o passo com deslocamento δ_S . S_r e S_c representam, respectivamente, o número de linhas e colunas de S . Os elementos da matriz que modela o passo são dados por

$$s_{ij} = \begin{cases} 1 & , \text{ for } ((i-1)\delta_S = j-1 \text{ e } i < S_r \text{ e } j < S_c) \text{ ou } (i = S_r \text{ e } j = S_c), \\ 0 & , \text{ caso contrário.} \end{cases} \quad (4.33)$$

- **Convolução** A matriz que modela a convolução, M , é definida como uma composição dos pesos (*i.e.*, do filtro), da dilatação, da correlação cruzada válida, do passo e do preenchimento com zeros, dada por

$$M = S (W D C) P = W (S (D C)^T P)^T, \quad (4.34)$$

sendo que $C = (c_{ijk})$ é o tensor de rank 3 que modela a correlação cruzada, de tamanho $((d_w - 1)\delta_D + 2) \times (d_x - (d_w - 1)\delta_D + 2\delta_P + 1) \times (d_x + 2\delta_P + 1)$, e dado por:

$$c_{ijk} = \begin{cases} 1 & , \text{ para } (i = k - j + 1 \text{ e } i < C_{\text{dim1}} \text{ e } j < C_{\text{dim2}} \text{ e } k < C_{\text{dim3}}) \\ & \text{ou } (i = C_{\text{dim1}} \text{ e } j = C_{\text{dim2}} \text{ e } k = C_{\text{dim3}}), \\ 0 & , \text{ caso contrário.} \end{cases} \quad (4.35)$$

- **Reamostragem por média** A matriz constante A modela a reamostragem por média e é computada como uma composição dos pesos, correlação cruzada válida, passo e preenchimento:

$$A = S (W C) P = W (S C^T P)^T, \quad (4.36)$$

onde $W = \left(\frac{1}{d_w}, \frac{1}{d_w}, \dots, \frac{1}{d_w}, 1\right)^T \in \mathbb{R}^{d_w+1}$ é um vetor constante de pesos e d_w é o tamanho do filtro.

- **Regularização por *dropout*** A matriz diagonal de tamanho $(d_x + 1) \times (d_x + 1)$

codifica o *dropout* como $R = (r_{i,j})$, sendo:

$$r_{ij} = \begin{cases} 1 & , \text{ para } i = j \text{ e } \text{RAND}() > \delta_R, \\ 0 & , \text{ caso contrário.} \end{cases} \quad (4.37)$$

Aqui, $\text{RAND}()$ é uma função que gera valores aleatórios uniformemente distribuídos sobre o intervalo $[0, 1]$, e δ_R é a probabilidade de um elemento passar a ter valor zero.

4.4 Abordagem Associativa para Redes Neurais Convolucionais

Ao longo desta Seção será apresentada a modelagem da composição das várias camadas lineares e de ativação apresentadas ao longo deste Capítulo, ao qual foi dado o nome de *ConformalLayers*.

Considere novamente a representação tensorial do ponto finito $X \in \mathbb{R}^{d+1}$ apresentada na Seção 4.1, cujas coordenadas representam a entrada de uma rede neural convolucional sequencial com uma coordenada extra. É sabido que é possível reescrever as camadas de redes neurais mais utilizadas como operações tensoriais (ver Seção 4.3). Essa reescrita, através da associatividade do produto de tensores, permite a composição das operações lineares de uma rede neural convolucional sequencial como

$$Z = UX, \quad (4.38)$$

sendo que U é obtido pela composição do conjunto de operações provenientes de uma camada linear da rede neural aplicada sobre o vetor X . A interpretação prática dessa equação é “o vetor Z representa um conjunto de transformações lineares U aplicadas sobre um vetor X ”. Substituindo X na Equação 4.15 por Z e reorganizando a equação de modo que U seja combinado com F_M e F_T , temos:

$$\begin{aligned} Y = \mathcal{L}(Z) &= (F_M + F_T Z) Z \\ &= (F_M + F_T (UX)) (UX) \\ &= (F_M U + (U^\top F_T^\top U)^\top X) X, \end{aligned} \quad (4.39)$$

onde \mathcal{L} é a *função da ConformalLayer*, uma sequência de operações lineares aplicadas a X seguidas pela aplicação da função de ativação ReSPro. Aqui, $^\top$ denota a transposição das duas primeiras dimensões dos tensores e a transposição da matriz.

Em redes neurais convolucionais sequenciais, é bastante comum intercalar as camadas lineares e as funções de ativação. Uma sequência de k ConformalLayers aplicadas a X pode ser escrita como

$$Y^{(k)} = \mathcal{L}^{(k)}(\mathcal{L}^{(k-1)}(\mathcal{L}^{(k-2)}(\dots))), \quad (4.40)$$

onde $\mathcal{L}^{(0)} = X$. Na notação $\mathcal{L}^{(l)}$, o índice sobrescrito indica a l -ésima camada das ConformalLayers, $X = (x'_1, x'_2, \dots, x'_{d_{\text{in}}}, x'_o)^\top \in \mathbb{R}^{d_{\text{in}}+1}$ representa o dado de entrada $x \in \mathbb{R}^{d_{\text{in}}}$, e $Y^{(k)} = (y_1^{(k)}, y_2^{(k)}, \dots, y_{d_{\text{out}}}^{(k)}, y_o^{(k)})^\top$ codifica a saída $y \in \mathbb{R}^{d_{\text{out}}^{(k)}}$, cujas coordenadas são dadas por $y_j = y_j^{(k)} / y_o^{(k)}$, para $j = \{1, 2, \dots, d_{\text{out}}^{(k)}\}$. Aqui, d_{in} e d_{out} representam, respectivamente, as dimensionalidades de entrada e saída de cada camada.

A utilização da expressão recursiva apresentada na Equação 4.40, no entanto, não é computacionalmente eficiente. Partindo de um valor $k = 3$ e então definindo para um valor k qualquer por indução, é possível reescrevê-la como

$$\begin{aligned} Y^{(3)} &= \mathcal{L}^{(3)} \left(\mathcal{L}^{(2)} \left(\mathcal{L}^{(1)}(X) \right) \right) \\ &= F_M^{(3)} U^{(3)} \left(F_M^{(2)} U^{(2)} F_M^{(1)} U^{(1)} X + \left(F_M^{(2)} U^{(2)} \left(U^{(1)\top} F_T^{(1)\top} U^{(1)} \right)^\top X \right) \right) \\ &\quad + \left(\left(U^{(1)\top} U^{(2)\top} F_T^{(2)\top} U^{(2)} U^{(1)} \right)^\top X \right) X \\ &\quad + \left(\left(U^{(3)\top} F_T^{(3)\top} U^{(3)} \right)^\top \left(F_M^{(2)} U^{(2)} F_M^{(1)} U^{(1)} X + \left(F_M^{(2)} U^{(2)} \left(U^{(1)\top} F_T^{(1)\top} U^{(1)} \right)^\top X \right) \right) \right) X \\ &\quad + \left(\left(U^{(1)\top} U^{(2)\top} F_T^{(2)\top} U^{(2)} U^{(1)} \right)^\top X \right) X \\ &\quad \left(F_M^{(2)} U^{(2)} F_M^{(1)} U^{(1)} X + \left(F_M^{(2)} U^{(2)} \left(U^{(1)\top} F_T^{(1)\top} U^{(1)} \right)^\top X \right) \right) X \\ &\quad + \left(\left(U^{(1)\top} U^{(2)\top} F_T^{(2)\top} U^{(2)} U^{(1)} \right)^\top X \right) X \\ Y^{(3)} &= F_M^{(3)} U^{(3)} F_M^{(2)} U^{(2)} F_M^{(1)} U^{(1)} X \\ &\quad + \left(F_M^{(3)} U^{(3)} F_M^{(2)} U^{(2)} \left(U^{(1)\top} F_T^{(1)\top} U^{(1)} \right)^\top X \right) X \\ &\quad + \left(F_M^{(3)} U^{(3)} \left(U^{(1)\top} U^{(2)\top} F_T^{(2)\top} U^{(2)} U^{(1)} \right)^\top X \right) X \\ &\quad + \left(\left(U^{(3)\top} F_T^{(3)\top} U^{(3)} \right)^\top F_M^{(2)} U^{(2)} F_M^{(1)} U^{(1)} X \right) F_M^{(2)} U^{(2)} F_M^{(1)} U^{(1)} X \end{aligned}$$

$$\begin{aligned}
Y^{(3)} &= F_M^{(3)} U^{(3)} F_M^{(2)} U^{(2)} F_M^{(1)} U^{(1)} X \\
&+ \left(F_M^{(3)} U^{(3)} F_M^{(2)} U^{(2)} \left(U^{(1)\top} F_T^{(1)\top} U^{(1)} \right)^\top X \right) X \\
&+ \left(F_M^{(3)} U^{(3)} \left(U^{(1)\top} U^{(2)\top} F_T^{(2)\top} U^{(2)} U^{(1)} \right)^\top X \right) X \\
&+ \left(\left(U^{(1)\top} U^{(2)\top} U^{(3)\top} F_T^{(3)\top} U^{(3)} U^{(2)} U^{(1)} \right)^\top X \right) X.
\end{aligned} \tag{4.41}$$

Por fim, a expressão da Equação 4.41 pode ser reescrita como

$$\begin{aligned}
Y^{(3)} &= \left(F_M^{(3)} U^{(3)} F_M^{(2)} U^{(2)} F_M^{(1)} U^{(1)} + \right. \\
&\quad \left(\sum_{l=1}^3 \left(F_M^{(3)} U^{(3)} F_M^{(2)} U^{(2)} \dots F_M^{(l+1)} U^{(l+1)} \right) \right. \\
&\quad \left. \left(U^{(1)\top} U^{(2)\top} \dots U^{(l)\top} F_T^{(l)\top} U^{(l)} \dots U^{(2)} U^{(1)} \right)^\top \right) X \\
Y^{(3)} &= \left(L_M^{(3)} + L_T^{(3)} X \right) X.
\end{aligned} \tag{4.42}$$

Assim, é possível observar que

$$Y^{(k)} = \left(L_M^{(k)} + L_T^{(k)} X \right) X, \tag{4.43}$$

sendo

$$L_M^{(k)} = F_M^{(k)} U^{(k)} F_M^{(k-1)} U^{(k-1)} \dots F_M^{(1)} U^{(1)} \tag{4.44}$$

uma matriz esparsa de tamanho $(d_{\text{out}}^{(k)} + 1) \times (d_{\text{in}} + 1)$, e

$$\begin{aligned}
L_T^{(k)} &= \sum_{l=1}^k \left(F_M^{(k)} U^{(k)} F_M^{(k-1)} U^{(k-1)} \dots F_M^{(l+1)} U^{(l+1)} \right) \\
&\quad \left(U^{(1)\top} U^{(2)\top} \dots U^{(l)\top} F_T^{(l)\top} U^{(l)} \dots U^{(2)} U^{(1)} \right)^\top
\end{aligned} \tag{4.45}$$

um tensor de rank 3 de tamanho $(d_{\text{out}}^{(k)} + 1) \times (d_{\text{in}} + 1) \times (d_{\text{in}} + 1)$ preenchido quase que em sua totalidade por zeros, exceto pela última fatia do tensor, que é uma matriz esparsa de tamanho $(d_{\text{in}} + 1) \times (d_{\text{in}} + 1)$. Uma observação importante é o fato de que os componentes $L_M^{(k)}$ e $L_T^{(k)}$ da Equação 4.43 **não dependem do vetor de entrada X , e que codificam a sequência completa de k ConformalLayers**. Assim, uma vez que os pesos tenham sido ajustados através do treinamento da rede neural convolucional (ver Equações 2.4 e 2.5), $L_M^{(k)}$ e $L_T^{(k)}$ podem ser calculados uma única vez utilizando as Equações 4.44 e 4.45, e posteriormente aplicados a quaisquer entradas X no processo de inferência de forma

computacionalmente eficiente.

4.5 Implementação das ConformalLayers

Esta Seção apresenta alguns detalhes da implementação das ConformalLayers, voltada à execução dos experimentos descritos no Capítulo 5. Essa implementação está disponível em <https://github.com/Prograf-UFF/ConformalLayers/> e é baseada no PyTorch 1.8, sobretudo pela melhoria dessa versão ao incluir o cálculo automático de derivadas parciais em operações envolvendo tensores esparsos. Assim, implementou-se as ConformalLayers como uma extensão da classe `nn.Module` do PyTorch 1.8.

O módulo `cl.ConformalLayers` atua como um módulo do tipo `nn.Sequential` e, em sua versão atual, aceita submódulos que imitam o comportamento dos módulos `nn.Conv1d`, `nn.Conv2d`, `nn.Conv3d`, `nn.AvgPool1d`, `nn.AvgPool2d`, `nn.AvgPool3d`, `nn.Dropout` e `nn.Flatten`. A configuração atual inclui como parâmetros, o passo, preenchimento com zeros e dilatação quando necessário. Além desses módulos, o módulo `cl.ReSPro` também é implementado. Perceba que, visando a facilidade da utilização da biblioteca por parte do usuário final, optou-se por manter os nomes e as assinaturas de métodos similares ao que já é utilizado atualmente no PyTorch, de modo que o mapeamento de uma biblioteca para a outra seja natural. A utilização dessa biblioteca em trechos de código podem ser visualizadas no Capítulo 5. A Tabela 4.1 sumariza os módulos implementados na versão atual da implementação das ConformalLayers.

A Figura 4.3 mostra o algoritmo implementado na função `forward`, definida no módulo `cl.ConformalLayers`. Durante o processo de treinamento, o módulo faz chamadas à implementação nativa dos submódulos PyTorch, quando existentes, para compor a rede neural (linha 5). As únicas exceções são o módulo `cl.ReSPro`, que é implementado de acordo com a Equação 4.15, e o cálculo do coeficiente $y_o'^{(k)}$ em todos os submódulos. Uma vez completado o processo de treinamento, o módulo `cl.ConformalLayers` constrói uma cache interna contendo a matriz esparsa $L_M^{(k)}$ (Equação 4.44) e a matriz esparsa representando a última fatia do tensor $L_T^{(k)}$ (Equação 4.45). É importante observar que essa cache precisa ser atualizada apenas se o usuário passar por novas iterações de ajuste de modelo (linhas 4, 9, e 10). Considerando que o valor de $y'^{(k)}$ pode ser diferente de 1 ao longo das camadas, o resultado da aplicação sequencial de submódulos deve ser dividido por $y'^{(k)}$ para obter a interpretação geométrica correta do vetor de saída (linhas 6 e 16). Devido à natureza esparsa de $L_M^{(k)}$ e $L_T^{(k)}$, a expressão na linha 15 é avaliada utilizando-se

Requer: *entrada*, uma representação tensorial dos dados de entrada

```

1:  $x \leftarrow \text{entrada}$  reformulada como um ponto com  $d_{\text{in}}$  coordenadas
2:  $x'_o \leftarrow \|x\|_2$ 
3: Se o módulo cl.ConformalLayers está em modo de treinamento faça
4:   Defina a cache como desatualizada
5:    $\text{saída}, y_o^{(k)} \leftarrow$  tensor de saída e coeficiente extra, resultados do processamento da entrada e de  $x'_o$ 
   utilizando a sequência de submódulos
6:    $\text{saída} \leftarrow \text{saída} / y_o^{(k)}$ 
7: Senão
8:   Se a cache está desatualizada faça
9:     Calcule e armazene  $L_M^{(k)}$  e  $L_T^{(k)}$  na cache
10:    Defina a cache como atualizada
11:   Senão
12:     Obtenha  $L_M^{(k)}$  e  $L_T^{(k)}$  da cache
13:   Fim Se
14:    $X \leftarrow (x_1, x_2, \dots, x_{d_{\text{in}}}, x'_o)^\top$ 
15:    $Y^{(k)} \leftarrow (L_M^{(k)} + L_T^{(k)})X$ 
16:    $y^{(k)} \leftarrow (y_1^{(k)}, y_2^{(k)}, \dots, y_{d_{\text{out}}}^{(k)}) / y_o^{(k)}$ 
17:    $\text{saída} \leftarrow y^{(k)}$  reformulada como o esperado para os dados de saída
18: Fim Se
19: retorne saída

```

Figura 4.3: A função `forward` do módulo `cl.ConformalLayers`.

apenas duas operações esparsas de multiplicação de matriz por vetor, um produto interno de vetores, uma adição e uma multiplicação.

Tabela 4.1: Módulos disponíveis na versão atual das ConformalLayers

Módulo	Descrição
Conv1d, Conv2d, Conv3d, ConvNd	Módulos de convolução implementada para sinais 1, 2, 3 e n -dimensionais
AvgPool1d, AvgPool2d, AvgPool3d, AvgPoolNd	Módulos de reamostragem por média implementada para sinais 1, 2, 3 e n -dimensionais
BaseActivation	Classe abstrata para extensão do módulo de camada de ativação
ReSPro	Módulo que corresponde à função de ativação ReSPro. A não-linearidade é controlada pelo parâmetro α , que pode ser passado como argumento ou calculado de maneira automática com base nos dados de entrada
Regularization	Na versão atual, apenas o <i>Dropout</i> está disponível. Nessa abordagem, durante o treinamento, existe uma probabilidade p , passada como argumento, de neurônios serem desligados

O cálculo dos tensores esparsos $U^{(l)}$, utilizados na obtenção de $L_M^{(k)}$ e $L_T^{(k)}$, é implementado com a biblioteca Minkowski Engine [49], uma vez que o PyTorch 1.8 não utiliza tensores esparsos como entrada para seus módulos nativos. Assim, a abordagem utilizada para converter uma sequência de operações lineares para uma matriz $U^{(l)}$ (ver Equação 4.39) é transformar um tensor identidade esparsa $I^{(l)}$ utilizando os módulos da Minkowski Engine devidamente adaptados para atuarem como módulos PyTorch.

O módulo `cl.ReSPro` permite ao usuário informar explicitamente o valor do parâmetro $\alpha^{(l)}$ como argumento durante a construção da arquitetura da rede ou deixar que o módulo `cl.ConformalLayers` estime o valor $\alpha^{(l)}$. Para isso, é necessário manter o registro da distância máxima entre a origem do espaço e o ponto resultante das operações lineares na l -ésima camada da rede. Antes da aplicação da matriz $U^{(l)}$ (que codifica as operações lineares), é razoável assumir que a norma L^2 da entrada da camada é 1, considerando que a coordenada x'_o possui, como valor, a distância euclidiana de x até a origem (linha 2 na Figura 4.3). Desse modo, o vetor de entrada X codifica um ponto que dista 1 unidade da origem e, por definição, o vetor resultante da aplicação da ReSPro na camada anterior (*i.e.*, $Y^{(l-1)}$, para $l > 1$) codifica um ponto que dista no máximo 1 unidade da origem. Após a aplicação de $U^{(l)}$, no entanto, não existe garantia de que essa distância máxima ainda seja de 1 unidade. Na prática, essa distância pode ser alterada de 1 para qualquer distância, dependendo da composição de $U^{(l)}$. Logo, a implementação apresentada deve tratar cada caso, ajustando a distância máxima como o limite superior que a norma L^2 pode assumir. Assim, $\alpha^{(l)}$ é calculado como o limite superior da distância entre os pontos que correspondem à saída da camada $(l - 1)$ com $U^{(l)}$ e a origem do espaço cartesiano.

Para os módulos `nn.AvgPool1d`, `nn.Dropout`, e `nn.Flatten`, o limite superior para a distância entre o ponto resultante e a origem do espaço é exatamente igual ao limite superior fornecido com a entrada uma vez que, nesses casos, a distância entre o ponto de saída e a origem é sempre menor ou igual à distância do ponto de entrada e a origem.

O módulo de `nn.Conv1d`, por outro lado, pode alterar a norma L^2 do vetor de saída para valores maiores em relação ao vetor de entrada. Assim, esse caso especial foi tratado com a utilização da desigualdade de Young [50], que define os limites superiores para o operador de convolução como:

$$\|g * h\|_r \leq \|g\|_p \|h\|_q, \text{ sujeito a } \frac{1}{p} + \frac{1}{q} = \frac{1}{r} + 1, \quad (4.46)$$

onde g e h se referem a sinais discretos e $\|\cdot\|_t$ denota a norma L^t . No caso apontado na implementação, utilizou-se $p = 2$, uma vez que a norma L^2 do vetor de entrada é conhe-

cida, $q = 1$, que corresponde à norma L^1 do filtro de convolução e, portanto, facilmente computável, e $r = 2$, de modo que se estime o limite superior da norma L^2 da saída. Com esses limites superiores, é possível manter registros de mudanças na distância dos pontos e atualizar $\alpha^{(l)}$ no módulo `cl.ReSPro` na l -ésima camada sempre que o processo de treinamento reajustar os pesos da rede neural convolucional. Todo esse processo de registro de alterações do valor de $\alpha^{(l)}$ é transparente para o usuário das ConformalLayers.

4.6 Discussão

Este capítulo explorou em detalhes a formulação que permite a modelagem da ReSPro como função de ativação associativa representada por tensores. Embora a modelagem da função seja enraizada em conceitos de álgebra geométrica utilizando o modelo conforme de geometria, esses conceitos acabam por serem atenuados face à composição tensorial da ReSPro, que utiliza apenas uma coordenada homogênea em sua representação. A escolha desse modelo se deve, sobretudo, ao maior controle que se tem sobre o intervalo de valores de saída da função de ativação, definidos pelo raio da hipersfera.

A composição da ReSPro permite a associatividade com as demais camadas tradicionalmente utilizadas em redes neurais convolucionais sequenciais. Assim, num segundo momento, se fez necessária a representação de camadas como convolução e reamostragem, por exemplo, num formato específico mais adequado à representação da ReSPro. Na prática, apesar dessas camadas já serem lineares e associativas, a forma como é utilizada tradicionalmente não permitia sua composição com a ReSPro. Assim, essas camadas foram reformuladas e apresentadas ao longo da Seção 4.3. O estudo e o desenvolvimento de outras funções de ativação com características similares à ReSPro está em andamento.

A associatividade da ReSPro com as demais camadas levou à composição das chamadas ConformalLayers. Essa composição, inicialmente apresentada como um aninhamento de chamadas recursivas, pôde ser expandida de modo que possuísse dois componentes principais e constantes: um tensor de rank 2 e um tensor de rank 3. Por serem constantes e, portanto, independentes da entrada, esses componentes podem ser pré-calculados após o treinamento, levando a um processo de inferência baseado em um número fixo de operações e independente da profundidade da rede. Os impactos na utilização dessa abordagem serão apresentados no Capítulo 5.

Foram apresentados, também, detalhes da implementação como uma biblioteca. Ao passo em que a porção baseada em PyTorch trata a definição da estrutura da biblioteca e

o do processo de treinamento, a porção que faz uso da Minkowski Engine trata de algumas operações sobre tensores esparsos. A forma como as ConformalLayers foram instrumentadas permite a utilização de componentes nativos do PyTorch durante o processo de ajuste dos pesos mas com os artefatos necessários às ConformalLayers. É o caso do parâmetro α , por exemplo, que precisa ser ajustado na saída de cada ConformalLayer, sobretudo pela característica de algumas camadas lineares moverem os pontos pelo espaço cartesiano em relação à origem. Além disso, os coeficientes associados à coordenada extra em cada camada também precisa ser gerenciado ao longo das camadas. Perceba que a modelagem do processo de treinamento foi mantida de maneira similar à abordagem tradicional. Isso se deve ao fato de que, para o treinamento, seriam necessárias várias atualizações dos tensores pré-calculados ao longo das épocas. Essa série de atualizações acabariam por deixar o treinamento mais custoso, mesmo levando-se em conta a quantidade reduzida de gradientes a serem calculados. Essa aparente limitação está atualmente em análise para posterior busca por soluções. Além disso, alguns módulos ainda não estão disponíveis na versão atual das ConformalLayers. A inclusão de módulos como camadas densas, por exemplo, é possível mas ainda não foi implementada na versão atual da biblioteca. Adicionalmente, o viés ou *bias* também não foi incluído na versão atual por demandar ajustes na modelagem. É possível modelar esse elemento em particular como um versor de translação no modelo conforme.

As **limitações** da abordagem dizem respeito a três itens: (i) a folga existente entre o valor α e a norma L^2 das entradas de cada camada, (ii) a ausência de camadas totalmente conectadas e (iii) a ausência de viés em camadas convolucionais. Ao passo em que o primeiro item se mostra o principal responsável pela acurácia aquém do esperado, o segundo item acaba por tornar o processo ligeiramente mais lento que o necessário, posto que sua modelagem como uma ConformalLayer poderia codificar a rede como uma composição de tensores associados do começo ao fim. O impacto do terceiro item sobre as características das ConformalLayers segue sob análise.

Este Capítulo apresentou, portanto, toda a composição das ConformalLayers, partindo da função de ativação, essencial ao aspecto associativo dessa arquitetura; passando pela remodelagem da porção linear e terminando na composição final em seus detalhes.

Capítulo 5

Experimentos e Resultados

Visando a validação da abordagem desenvolvida, foram desenhados experimentos que avaliassem as ConformalLayers quanto à acurácia na tarefa de classificação de imagens, consumo de memória e tempo de inferência, quando comparados à abordagem convencional.

Os experimentos relacionados à acurácia foram executados em uma máquina com processador Intel Xeon E5-2698 v4 de 2.2GHz com 512Gb de memória RAM e 8 GPUs NVIDIA Tesla P100-SXM2 (Arquitetura Pascal) com 16Gb de memória cada. Os experimentos relacionados ao consumo de memória e tempo de inferência foram executados em uma máquina com Intel Core i7-4770 CPU com 3.4GHz com 20Gb de RAM e uma GPU NVIDIA GTX 1050 Ti (Arquitetura Pascal) com 4Gb de memória. Todos os experimentos foram executados dentro de um container Docker [51]. Para evitar a possível distribuição de algumas subtarefas do treinamento das redes neurais em ambientes com múltiplas GPUs, utilizou-se a opção `NVIDIA_VISIBLE_DEVICES` para permitir que apenas uma única GPU fosse visível ao container.

As comparações entre arquiteturas foram feitas duas a duas, de modo que as arquiteturas com sufixo **CL** são baseadas em ConformalLayers e, portanto, associativas. Para experimentos visando a verificação acurácia em redes baseadas em ConformalLayers foram utilizados três bases de dados, bastante conhecidas na literatura: MNIST [52], que consiste em dez classes com imagens de dígitos manuscritos; FashionMNIST [53], composta por dez classes de imagens de itens de vestuário e CIFAR10 [54], composta por dez classes de imagens de veículos de transporte e animais. Tanto a base MNIST quanto a base FashionMNIST possuem em imagens em escala de cinza (com apenas um canal de cor) de tamanho 27×27 , ao passo em que a base CIFAR10 consiste em imagens com três canais de cor e de tamanho 32×32 . Assim, tendo por objetivo a garantia da utilização

Tabela 5.1: Espaço de busca para os hiperparâmetros utilizados nos experimentos.

	Limite inferior	Limite Superior	Tipo
Tamanho do Lote	2048	4096	Distribuição uniforme discretizada em passos unitários
Épocas	10	50	Distribuição uniforme discretizada em passos unitários
Taxa de Aprendizado	0.001	1.0	Distribuição uniforme contínua
Otimizador	{Adam, RMSprop}		Distribuição categórica

da exata arquitetura de rede para todas as bases de dados, de modo que não fossem usados mais parâmetros em um ou outro experimento, optou-se pelo ajuste dos dados nas bases MNIST e FashionMNIST, em que as imagens passaram a ter três canais, obtidos através da cópia do canal original e, por meio do preenchimento com zeros nas bordas das imagens, passaram a ter tamanho 32×32 . Assim, ao final do processo, todas as bases utilizadas na avaliação da acurácia possuíam imagens com três canais de cor e tamanho 32×32 . Nos experimentos que visavam a comparação quanto à utilização de memória e tempo de inferência, em que foram necessárias bases de dados maiores e que a capacidade de classificação não era o foco principal, foram criadas bases de dados contendo imagens geradas aleatoriamente. Essas base de dados sintéticas possuem, também, imagens com três canais de cor e tamanho 32×32 .

No contexto dos experimentos para avaliação da acurácia de modelos baseados em ConformalLayers, foi feita a varredura de hiperparâmetros objetivando a seleção de bons modelos para que o comparativo fosse justo. Os limites do espaço de varredura estão definidos na Tabela 5.1. Assim, utilizou-se uma abordagem Bayesiana para varredura no espaço de parâmetros, assumindo a acurácia obtida no processo de validação como métrica e Hyperband [56], configurado com $\text{max_iter} = 50$, $s = 2$, e $\eta = 3$, como critério de parada. A Tabela 5.2 elenca os parâmetros selecionados para cada par definido pela arquitetura e pela base de dados. Os conjuntos de hiperparâmetros que levaram às top-10 acurácias estão disponíveis no Apêndice A. Em todos os experimentos, considerou-se os valores de α utilizados pela ReSPro definidos de maneira automática com base nos dados.

5.1 Experimento I - *Baseline* Linear

O objetivo geral desse experimento é a comparação da acurácia de arquiteturas com uma função de ativação puramente linear, cuja arquitetura foi denominada **BaseLinearNet**, apresentada com sua respectiva implementação na Figura 5.1; uma função de ativação não-linear amplamente adotada na literatura, nesse caso a ReLU, cuja arquitetura foi denominada **BaseReLUNet** e está representada com sua implementação na Figura 5.2; e uma

Tabela 5.2: Hiperparâmetros selecionados para cada arquitetura e base de dados utilizando uma abordagem Bayesiana implementada pela ferramenta *Weights and Biases* [55]. Para as redes **LeNet** e **LeNetCL**, o número de épocas foi mantido fixo em 200 durante o treinamento, uma vez que os intervalos mostrados na Tabela 5.1 se mostraram insuficientes para o ajuste dos modelos.

BaseLinearNet	MNIST	FashionMNIST	CIFAR10
Tamanho do Lote	2868	3111	2198
Épocas	44	15	41
Taxa de Aprendizado	0.02039	0.05305	0.09827
Otimizador	Adam	Adam	Adam
BaseReLUNet	MNIST	FashionMNIST	CIFAR10
Tamanho do Lote	2838	3277	3677
Épocas	50	48	21
Taxa de Aprendizado	0.090331	0.08607	0.01
Otimizador	Adam	Adam	RMSprop
BaseReSProNet	MNIST	FashionMNIST	CIFAR10
Tamanho do Lote	2429	2339	3056
Épocas	50	45	19
Taxa de Aprendizado	0.7699	0.5601	0.4542
Otimizador	Adam	Adam	Adam
LeNet	MNIST	FashionMNIST	CIFAR10
Tamanho do Lote	2979	2444	2317
Épocas	200	200	200
Taxa de Aprendizado	0.004722	0.012	0.00159
Otimizador	RMSprop	Adam	RMSprop
LeNetCL	MNIST	FashionMNIST	CIFAR10
Tamanho do Lote	2088	2763	3595
Épocas	200	200	200
Taxa de Aprendizado	0.02	0.02302	0.01991
Otimizador	Adam	Adam	Adam

arquitetura implementada com a ReSPro (**BaseReSProNet**), representada na Figura 5.3.

A utilização de redes neurais convolucionais simples, contendo apenas uma camada cada, é justificada pela ideia de que à medida que a profundidade da rede aumenta, sob certos aspectos, aumenta também a capacidade de aprendizado da rede [2]. Ao utilizar uma rede com apenas uma camada, procurou-se evitar esse tipo de viés no experimento. As três arquiteturas esperam imagens de tamanho 32×32 em três canais como entrada e incluem apenas uma porção para extração de características (camada convolucional) e uma camada totalmente conectada com *bias* para classificar as características em uma das 10 classes definidas pela base de dados MNIST, FashionMNIST e CIFAR10.

A porção convolucional da rede **BaseLinearNet** é composta por uma camada convolucional com *bias* seguido por uma camada de reamostragem por valor médio. Perceba que essa arquitetura possui apenas camadas lineares. A arquitetura **BaseReLUNet** es-

tende a `BaseLinearNet` ao incluir uma função de ativação ReLU na saída da camada de convolução, ambas implementadas com funções e objetos nativos do PyTorch.

```

1 import torch.nn as nn
2
3 class BaseLinearNet(nn.Module):
4     def __init__(self) -> None:
5         super(BaseLinearNet, self).__init__()
6         self.features = nn.Sequential(
7             nn.Conv2d(3, 32, kernel_size=3),
8             nn.AvgPool2d(kernel_size=2, stride=2),
9         )
10        self.fc = nn.Linear(7200, 10)
11
12    def forward(self, x):
13        x = self.features(x)
14        x = torch.flatten(x, 1)
15        x = self.fc(x)
16        return x

```

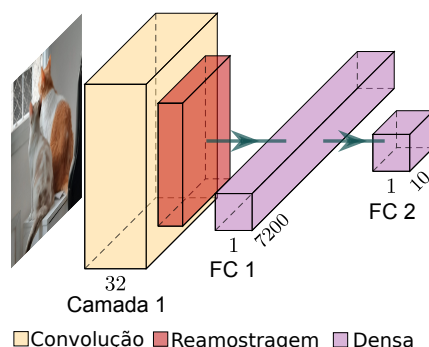


Figura 5.1: Implementação e arquitetura da rede `BaseLinearNet`, definida por uma camada convolucional com *bias*, um camada de reamostragem por média e uma camada totalmente conectada (com *bias*) com saída para as dez classes de cada base de dados. A ausência de ativações não-lineares torna a rede totalmente linear, limitando a capacidade de aprendizado da rede mas estabelecendo um bom *baseline* de comparação.

```

1 import torch.nn as nn
2
3 class BaseReLUNet(nn.Module):
4     def __init__(self) -> None:
5         super(BaseReLUNet, self).__init__()
6         self.features = nn.Sequential(
7             nn.Conv2d(3, 32, kernel_size=3),
8             nn.ReLU(),
9             nn.AvgPool2d(kernel_size=2, stride=2),
10        )
11        self.fc = nn.Linear(7200, 10)
12
13    def forward(self, x):
14        x = self.features(x)
15        x = torch.flatten(x, 1)
16        x = self.fc(x)
17        return x

```

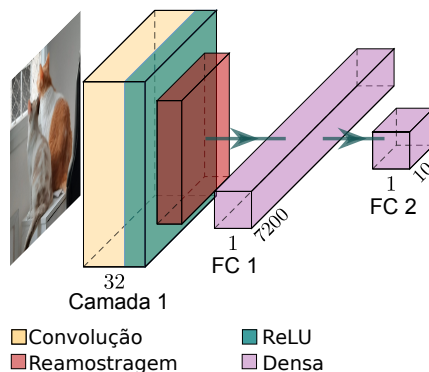


Figura 5.2: Implementação e arquitetura da rede `BaseReLUNet`, definida por uma camada convolucional com *bias* seguida por uma ativação do tipo ReLU, uma camada de reamostragem por média e uma camada totalmente conectada (com *bias*) com saída para as dez classes de cada base de dados.

```

1 import cl
2 import torch.nn as nn
3
4 class BaseReSProNet(nn.Module):
5     def __init__(self) -> None:
6         super(BaseReSProNet, self).__init__()
7         self.features = cl.ConformalLayers(
8             cl.Conv2d(3, 32, kernel_size=3),
9             cl.ReSPro(),
10            cl.AvgPool2d(kernel_size=2, stride=2),
11        )
12        self.fc = nn.Linear(7200, 10)
13
14    def forward(self, x) -> torch.Tensor:
15        x = self.features(x)
16        x = torch.flatten(x, 1)
17        x = self.fc(x)
18        return x

```

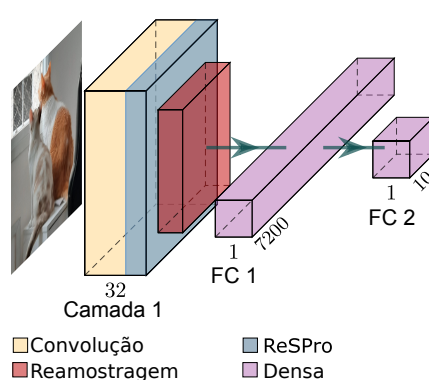


Figura 5.3: Implementação e arquitetura da rede **BaseReSProNet**, definida por uma camada convolucional sem *bias* seguida por uma ativação do tipo ReSPro, uma camada de reamostragem por média e uma camada totalmente conectada (com *bias*) com saída para as dez classes de cada base de dados.

A arquitetura **BaseReSProNet**, por outro lado, foi implementada utilizando **ConformalLayers** e inclui uma camada convolucional sem *bias*, a função ReSPro como função de ativação na saída da convolução e uma camada de reamostragem por valor médio. Os filtros de convolução em todas as CNNs possuem tamanho 3×3 , 32 canais de saída, sem preenchimento e passo de um pixel. Os filtros de reamostragem possuem tamanho 2×2 , sem preenchimento e passo de um pixel.

Os resultados da classificação estão apresentados na Tabela 5.3. A arquitetura **BaseReSProNet** possui uma acurácia maior quando comparada com a **BaseLinearNet**, com um aumento variando de 1.36 a 2.17 pontos percentuais, dependendo da base de dados, e menor acurácia quando comparada com a **BaseReLUNet**, com uma diminuição de 1.95 a 7.61 pontos percentuais. A comparação com a **BaseLinearNet** reforça que ideia de que a função de ativação apresentada torna a **BaseReSProNet** mais interessante do que uma simples abordagem linear. O parâmetro α da ReSPro, se bem definido, garante a não-linearidade necessária ao aprendizado do modelo. A utilização de um valor α muito maior que a norma L^2 esperada para os dados de entrada leva a ReSPro a se comportar de maneira mais similar a um mapeamento linear. Esse experimento mostra que a estratégia adotada para estimativa automática do valor de α impede que a **BaseReSProNet** se comporte como a **BaseLinearNet**. É importante ressaltar que existe uma ordenação quanto à complexidade das bases de dados utilizadas: a base CIFAR10 é mais complexa que a base FashionMNIST que, por sua vez, é mais complexa que a base MNIST. Assim, conforme a complexidade da base de dados aumenta, como mostra a comparação dos dados obtidos da base MNIST com os dados obtidos da FashionMNIST com os da CIFAR10, é perceptível que a melhoria da acurácia da **BaseReSProNet** decresce rapidamente quando comparada com a **BaseReLUNet**. A função ReLU parece prover uma melhor acurácia quando comparada com a ReSPro. Essa diferença sugere uma desvantagem na implementação de redes neurais menos profundas utilizando ReSPro quando se trata de bases de dados mais complexas.

Tabela 5.3: Acurácia da validação das CNNs aplicadas à MNIST, FashionMNIST e CIFAR10 para o Experimento I.

	MNIST	FashionMNIST	CIFAR10
BaseLinearNet	92.12%	82.19%	40.50%
BaseReLUNet	97.24%	85.90%	49.47%
BaseReSProNet	94.29%	84.01%	41.86%

5.2 Experimento II - LeNet \times LeNetCL

Ao passo em que o Experimento I visava estabelecer a posição das ConformalLayers em relação às abordagens convencionais em termos de acurácia de redes simples com apenas uma camada de ativação, o Experimento II visa avaliar a acurácia em redes um pouco mais profundas, em que existe o encadeamento de múltiplas camadas. Com esse objetivo, utilizou-se arquiteturas baseadas na LeNet-5 [57]. A escolha dessa arquitetura se deve, sobretudo, ao seu formato compacto e de fácil entendimento ao passo em que é capaz de prover resultados mais interessantes, dada sua maior capacidade de aprendizado quando comparadas às redes de *baseline* apresentadas na Seção 5.1.

A arquitetura **LeNet** consiste em camadas configuradas com muitas semelhanças em relação à arquitetura original. As diferenças residem em três pontos: (i) na substituição da reamostragem por valor máximo (*max pooling*) por reamostragem por média (*average pooling*); (ii) na ausência de *bias* nas convoluções e (iii) na utilização da função de ativação ReLU ao invés da função logística.

A arquitetura **LeNetCL** implementa a porção de extração de características (porção convolucional) utilizando ConformalLayers. Assim, o *bias* também não é utilizado nas convoluções, a função ReSPro é utilizada como função de ativação e a estratégia de reamostragem é a por média.

As Figuras 5.4 e 5.5 apresentam as implementações das arquiteturas **LeNet** e **LeNetCL**, respectivamente, utilizadas nesse Experimento e baseadas na LeNet-5, ao passo em que a Figura 5.6 apresenta as arquiteturas em si. As diferenças entre as arquiteturas residem, sobretudo, na porção convolucional, em que a **LeNet** é implementada com PyTorch puro da maneira convencional ao passo em que a **LeNetCL** é implementada com ConformalLayers com ReSPro como função de ativação sendo, portanto, associativa.

Tabela 5.4: Acurácia da validação para arquiteturas de redes neurais baseadas na LeNet-5 para as bases de dados MNIST, FashionMNIST e CIFAR10.

	MNIST	FashionMNIST	CIFAR10
LeNet	98.87%	90.13%	59.05%
LeNetCL	97.33%	89.12%	53.27%

```

1 import torch.nn as nn
2 import torch.nn.functional as func
3 import cl
4
5 class LeNetCL(nn.Module):
6     def __init__(self):
7         super(LeNetCL, self).__init__()
8         self.features = cl.ConformalLayers(
9             cl.Conv2d(3, 6, kernel_size=5),
10            cl.ReSPro(),
11            cl.AvgPool2d(kernel_size=2, stride=2),
12            cl.Conv2d(6, 16, kernel_size=5),
13            cl.ReSPro(),
14            cl.AvgPool2d(kernel_size=2, stride=2),
15        )
16        self.fc1 = nn.Linear(16*5*5, 120)
17        self.fc2 = nn.Linear(120, 84)
18        self.fc3 = nn.Linear(84, 10)
19
20    def forward(self, x):
21        x = self.features(x)
22        x = x.view(x.shape[0], -1)
23        x = func.relu(self.fc1(x))
24        x = func.relu(self.fc2(x))
25        x = self.fc3(x)
26        return x

```

Figura 5.4: Implementação da arquitetura LeNetCL, utilizada no Experimento II, e definida utilizando-se os módulos das ConformalLayers. Toda a porção convolucional é computada de maneira associativa.

```

1 import torch.nn as nn
2 import torch.nn.functional as func
3
4 class LeNet(nn.Module):
5     def __init__(self):
6         super(LeNet, self).__init__()
7         self.features = nn.Sequential(
8             nn.Conv2d(3, 6, kernel_size=5),
9             nn.ReLU(),
10            nn.AvgPool2d(kernel_size=2, stride=2),
11            nn.Conv2d(6, 16, kernel_size=5),
12            nn.ReLU(),
13            nn.AvgPool2d(kernel_size=2, stride=2),
14        )
15        self.fc1 = nn.Linear(16*5*5, 120)
16        self.fc2 = nn.Linear(120, 84)
17        self.fc3 = nn.Linear(84, 10)
18
19    def forward(self, x):
20        x = self.features(x)
21        x = x.view(x.shape[0], -1)
22        x = func.relu(self.fc1(x))
23        x = func.relu(self.fc2(x))
24        x = self.fc3(x)
25        return x

```

Figura 5.5: Implementação da arquitetura LeNet, utilizada no Experimento II, e definida utilizando-se os módulos nativos do PyTorch.

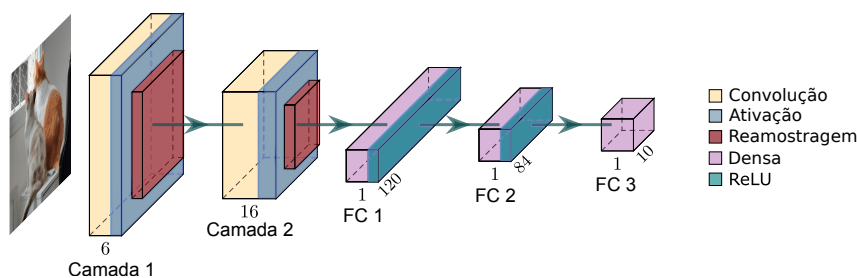


Figura 5.6: Arquiteturas LeNetCL e LeNet, cujas implementações estão descritas nas Figuras 5.4 e 5.5, respectivamente. As diferenças entre as duas arquiteturas residem na porção convolucional da rede, *i.e.*, camadas 1 e 2.

Como é possível notar na Tabela 5.4, a arquitetura **LeNetCL** registrou uma melhoria significativa quando comparada com a acurácia da **BaseReSProNet**, registrada na Tabela 5.3. Isso se deve, sobretudo, à maior profundidade da rede, o que aumenta consideravelmente sua capacidade de modelar dados mais complexos. Além disso, as diferenças na acurácia de classificação obtida com a **LeNet** e **LeNetCL** são menores que as diferenças entre a **BaseReLU** e **BaseReSProNet** para todas as bases de dados, variando de 1,01 (FashionMNIST) a 5,78 (CIFAR10) pontos percentuais.

A redução relativa obtida ao compararmos a acurácia das redes neurais convolucionais construídas com ReSPro e ReLU se devem, sobretudo, à diferença entre o valor do parâmetro α e o limite superior definido pela norma L^2 (ver Equação 4.46). Essa acurácia ligeiramente menor pode ser aceitável em cenários em que o consumo de memória e o custo computacional associado às redes neurais convolucionais tradicionais são fatores limitantes de suas aplicações. Os próximos experimentos, detalhados nas Seções 5.3 e 5.4, analisam justamente as vantagens em termos de performance computacional das **ConformalLayers** durante o processo de inferência.

5.3 Experimento III - Profundidade da Rede \times Tempo de Inferência

No primeiro experimento para avaliação de performance computacional em termos de tempo de inferência e memória necessária, comparou-se o impacto da profundidade da rede neural convolucional de redes implementadas da maneira convencional e utilizando-se **ConformalLayers**. A Equação 4.43, por sua característica de requerer uma quantidade fixa de memória e de operações aritméticas em sua execução, motivou a ideia de que o tempo de inferência de redes neurais convolucionais implementadas com **ConformalLayers** seria fixo, independente da profundidade, *i.e.*, do número de camadas da rede. Esse experimento é desenhado especificamente para a validação dessa ideia.

Para a execução desse experimento foram criados aleatoriamente lotes (batches) compostos por 64 imagens com três canais de tamanho 32×32 pixels, cujas intensidades foram sorteadas com base numa distribuição uniforme. Esses lotes de imagens foram utilizados como entrada da rede nesse experimento. Adicionalmente, os pesos das redes neurais convolucionais também foram gerados de maneira aleatória com base numa distribuição uniforme e os mesmos não foram ajustados por treinamento, uma vez que o objetivo desse experimento é a medição de tempo de inferência da solução implementada

e não sua capacidade de classificação.

```

1 import torch.nn as nn
2 import cl
3
4 class DkNetCL(nn.Module):
5     def __init__(self, depth):
6         super(DkNetCL, self).__init__()
7         input_channels = 3
8         filters = 32
9         L = []
10        for i in range(depth):
11            L.append(cl.Conv2d(input_channels, filters, 3, padding=1))
12            L.append(cl.ReSPro())
13            input_channels = filters
14        self.features = cl.ConformalLayers(*L)
15        self.fc1 = nn.Linear(32768, 10)
16
17    def forward(self, x):
18        x = self.features(x)
19        x = x.view(x.shape[0], -1)
20        x = self.fc1(x)
21        return x

```

Figura 5.7: Implementação da arquitetura DkNetCL, utilizada no Experimento III, e definida utilizando-se os módulos das ConformalLayers. Toda a porção convolucional é computada de maneira associativa na linha 14.

```

1 import torch.nn as nn
2
3 class DkNet(nn.Module):
4     def __init__(self, depth):
5         super(DkNet, self).__init__()
6         input_channels = 3
7         filters = 32
8         L = []
9         for i in range(depth):
10            L.append(nn.Conv2d(input_channels, filters, 3, padding=1))
11            L.append(nn.ReLU())
12            input_channels = filters
13        self.features = nn.Sequential(*L)
14        self.fc1 = nn.Linear(32768, 10)
15
16    def forward(self, x):
17        x = self.features(x)
18        x = x.view(x.shape[0], -1)
19        x = self.fc1(x)
20        return x

```

Figura 5.8: Implementação da arquitetura DkNet, utilizada no Experimento III, e definida utilizando-se os módulos nativos do PyTorch.

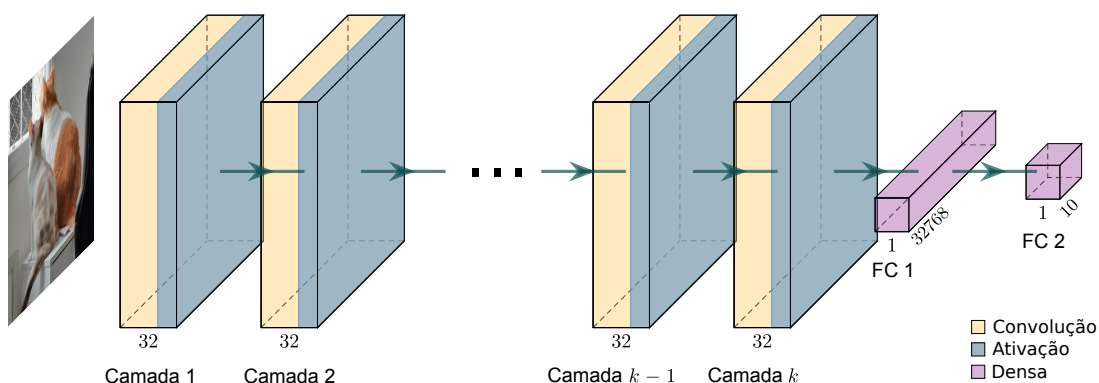


Figura 5.9: Arquitetura das redes DkNetCL e DkNet, cujas implementações são apresentadas nas Figuras 5.7 e 5.8, respectivamente.

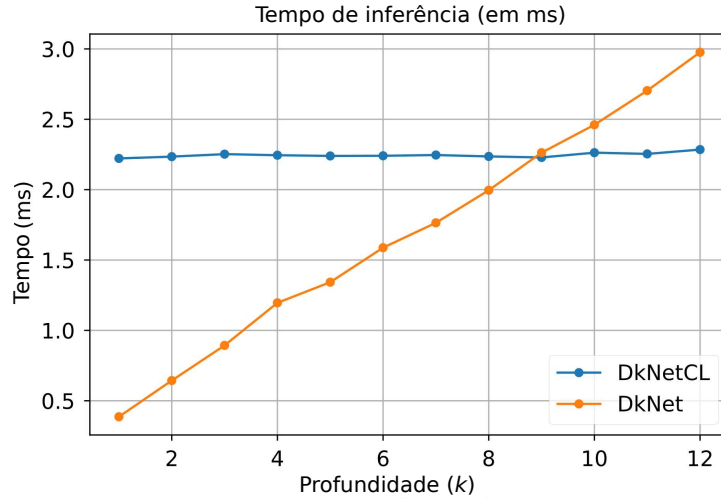


Figura 5.10: Tempos de inferência para as arquiteturas DkNetCL e DkNet em função da profundidade da rede (k). Perceba o comportamento independente da profundidade da rede neural convolucional implementada utilizando ConformalLayers.

As Figuras 5.7 e 5.8 apresentam a implementação das redes neurais DkNetCL e DkNet, respectivamente, ao passo em que a Figura 5.9 ilustra a arquitetura de ambas as redes, utilizadas nesse experimento. Essa arquitetura possui sequencias de camadas convolucionais com filtros de tamanho 3×3 , com 32 canais de saída, passo de 1 unidade, e preenchimento de 1 unidade. Tal configuração permite manter a entrada e a saída da porção convolucional do mesmo tamanho. Após a porção convolucional são incluídas duas camadas totalmente conectadas, com *bias*, e que produzem vetores com 32.768 e 10 componentes, respectivamente, e sem funções de ativação. O número de camadas convolucionais é controlado pela variável k . Assim, uma arquitetura criada com $k = 3$, por exemplo, possui três camadas convolucionais antes da porção totalmente conectada. As diferenças entre a DkNetCL e a DkNet residem no fato de que a primeira utiliza ReSPro como função de ativação, sem *bias* nas convoluções e implementa a porção convolucional com ConformalLayers sendo, portanto, associativa. A segunda, por sua vez, utiliza módulos nativos do PyTorch, ReLU como função de ativação e *bias* nos módulos de convolução sendo, portanto, não associativa.

Os tempos de inferência médios de 100 execuções das arquiteturas DkNetCL e DkNet para os diferentes valores de k , *i.e.*, para diferentes profundidades, estão apresentados na Figura 5.10. A primeira observação importante se baseia no fato de que os tempos de inferência associados à DkNetCL são constantes conforme a profundidade da rede aumentar. Para a DkNet, por outro lado, é possível observar um tempo linear monotonicamente crescente conforme a profundidade da rede aumenta. Para $k = 1$, o tempo de inferência da DkNet é quase 6 vezes menor que o tempo da DkNetCL. Essa aparente desvantagem,

no entanto, é rapidamente superada conforme as redes se tornam mais profundas. Para $k \geq 9$, a rede neural convolucional baseada nas ConformalLayers é mais vantajosa do que a rede neural com camadas não associativas.

A desvantagem inicial da **DkNetCL** em relação à **DkNet** se deve, sobretudo, à ausência de bibliotecas eficientes para tratar tensores esparsos em GPUs. Bibliotecas otimizadas para esse tipo de tensor contribuiriam consideravelmente para a redução dessa desvantagem. Além disso, arquiteturas mais modernas de GPUs, como a Ampere, se mostram bastante otimizados para estruturas de dados esparsas, como mostrado por Anzt *et al.* [58].

5.4 Experimento IV - Tamanho do Lote \times Tempo de Inferência

O segundo experimento relacionado à performance de implementações baseadas em ConformalLayers busca dar suporte à ideia de que, uma vez que são executadas menos operações no processo de inferência, o processo deve ser mais rápido que a abordagem tradicional. Além disso, considerando que não há necessidade de manter dados intermediários e, assim, ocupar menos memória, poderia executar a inferência em mais imagens simultaneamente, *i.e.*, em lotes maiores.

Para esse experimento, foram modeladas duas novas arquiteturas bastante similares: **D3ModNetCL**, cuja implementação está na Figura 5.11 e **D3ModNet**, cuja implementação está na Figura 5.12, sendo a primeira implementada com as camadas associativas provenientes das ConformalLayers e a segunda implementada com PyTorch nativo e, portanto, não associativo. Ambas as arquiteturas estão representadas na Figura 5.13. As principais diferenças em relação às arquiteturas de rede neural convolucional apresentadas no Experimento III (Seção 5.3) são: (i) agora as redes possuem profundidade fixa em 3; (ii) fazem reamostragem por média de tamanho 2×2 após a função de ativação; e (iii) tanto a reamostragem quanto a convolução não possuem preenchimento. A rede **D3ModNetCL** utiliza ReSPro como função de ativação, tornando-a associativa e não utiliza *bias* na convolução, ao passo em que a **D3ModNet** utiliza ReLU como função de ativação e *bias* na convolução.


```

1 import torch.nn as nn
2 import cl
3
4 class D3ModNetCL(nn.Module):
5     def __init__(self):
6         super(D3ModNetCL, self).__init__()
7         self.features = cl.ConformalLayers(
8             cl.Conv2d(in_channels=3, out_channels=32, kernel_size=3),
9             cl.ReSPro(),
10            cl.AvgPool2d(kernel_size=2, stride=2),
11            cl.Conv2d(in_channels=32, out_channels=32, kernel_size=3),
12            cl.ReSPro(),
13            cl.AvgPool2d(kernel_size=2, stride=2),
14            cl.Conv2d(in_channels=32, out_channels=32, kernel_size=3),
15            cl.ReSPro(),
16            cl.AvgPool2d(kernel_size=2, stride=2),
17        )
18        self.fc1 = nn.Linear(128, 10)
19
20     def forward(self, x):
21         x = self.features(x)
22         x = x.view(x.shape[0], -1)
23         x = self.fc1(x)
24         return x

```

Figura 5.11: Implementação da arquitetura D3ModNetCL, utilizada no Experimento IV, e definida utilizando-se os módulos das ConformalLayers. Toda a porção convolucional é computada de maneira associativa.

```

1 import torch.nn as nn
2
3 class D3ModNet(nn.Module):
4     def __init__(self):
5         super(D3ModNet, self).__init__()
6         self.features = nn.Sequential(
7             nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3),
8             nn.ReLU(),
9             nn.AvgPool2d(kernel_size=2, stride=2),
10            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3),
11            nn.ReLU(),
12            nn.AvgPool2d(kernel_size=2, stride=2),
13            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3),
14            nn.ReLU(),
15            nn.AvgPool2d(kernel_size=2, stride=2),
16        )
17        self.fc1 = nn.Linear(128, 10)
18
19     def forward(self, x):
20         x = self.features(x)
21         x = x.view(x.shape[0], -1)
22         x = self.fc1(x)
23         return x

```

Figura 5.12: Implementação da arquitetura D3ModNet, utilizada no Experimento IV, e definida utilizando-se os módulos nativos do PyTorch.

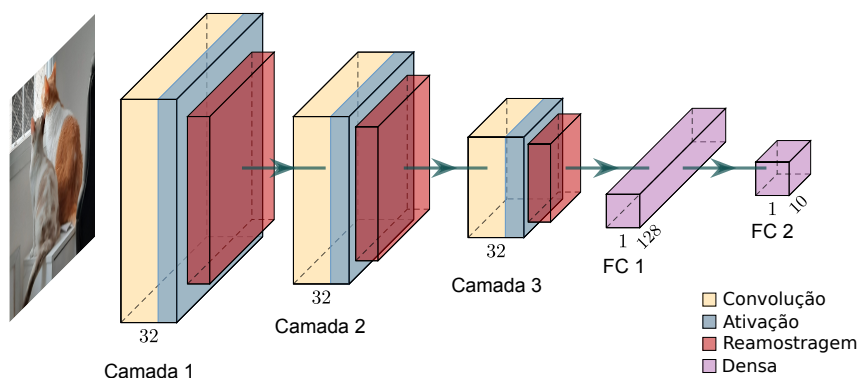


Figura 5.13: Arquitetura das redes D3ModNetCL e D3ModNet, cujas implementações são apresentadas nas Figuras 5.11 e 5.12, respectivamente.

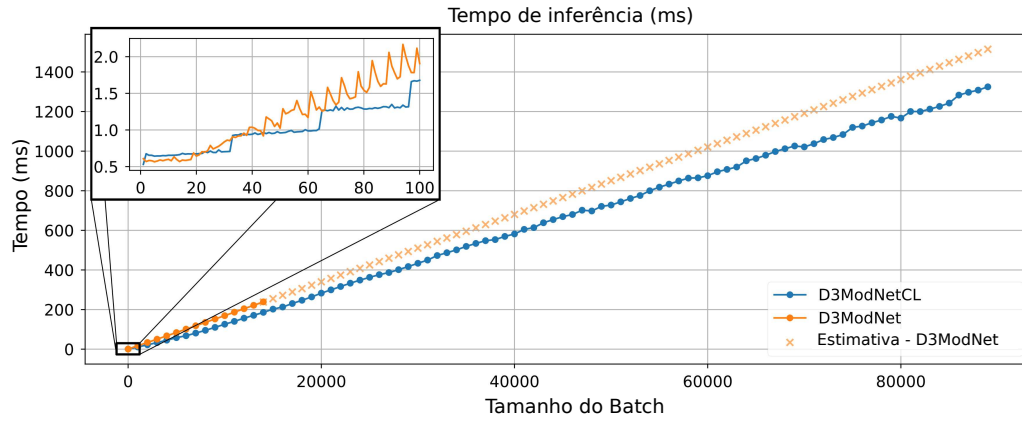


Figura 5.14: Tempo de inferência para os modelos D3ModNetCL e D3ModNet, implementados com a abordagem associativa e a abordagem tradicional, respectivamente, considerando a variação no tamanho do lote. Por limitações de memória, não foi possível executar a D3ModNet para lotes maiores que 14 mil imagens. Assim, um regressor linear foi utilizado para a estimativa do tempo para essa abordagem. Em destaque, o comportamento incremental de memória do tempo de inferência mesmo em lotes pequenos.

Os tempos médios de 100 inferências para cada arquitetura de rede neural convolucional estão dispostos na Figura 5.14. A primeira observação a ser feita é que a arquitetura D3ModNetCL é capaz de executar a inferência mais rapidamente que a D3ModNet. Isso se deve, sobretudo, à quantidade reduzida de operações feitas ao utilizar as camadas associativas implementadas nas ConformalLayers. Uma análise mais minuciosa da porção ampliada revela um comportamento interessante: o tempo de inferência da D3ModNetCL é incrementado em passos, onde cada passo possui um comprimento de 32 lotes. Esse comportamento está intimamente relacionado ao tamanho do bloco de *threads* (esse tamanho de bloco também é conhecido como *warp*) da GPU utilizada nesse experimento.

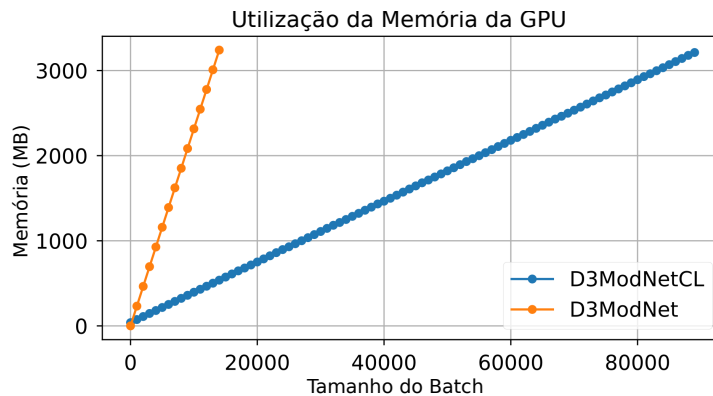


Figura 5.15: Consumo de memória da GPU durante o processo de inferência das arquiteturas D3ModNetCL e D3ModNet, implementados com a abordagem associativa e a abordagem tradicional, respectivamente, considerando a variação no tamanho do lote. Perceba que a memória durante a execução da D3ModNet é rapidamente preenchida, inviabilizando a inferência para lotes maiores que 14 mil imagens.

A GPU NVIDIA GTX 1050 Ti possui um bloco de threads com 32 threads. Na prática o tamanho b de blocos de thread define o número de ciclos necessários para executar algum cálculo. Assim, para m operações, a GPU necessita de $\lceil m/b \rceil$ ciclos, onde $\lceil \cdot \rceil$ representa a função teto.

Outra observação interessante pode ser obtida com base na análise da Figura 5.14 juntamente com a Figura 5.15. A GPU utilizada nesses experimentos possui 4GB de memória e, como é possível notar na Figura 5.15, o consumo de memória na arquitetura **D3ModNet** possui uma curva mais inclinada do que a curva associada ao consumo pela arquitetura **D3ModNetCL**. Em termos práticos, isso indica que a memória da GPU é consumida mais rapidamente em abordagens não-associativas. Adicionalmente, o modelo implementado com **ConformalLayers** suporta lotes consideravelmente maiores antes de consumir toda a memória. Isso ocorre, sobretudo, pelo fato do mapa de características resultante de cada operação da **D3ModNet** ter que ser armazenado na memória para ser passado como entrada para a operação seguinte. Assim, a curva associada a **D3ModNet** na Figura 5.14 mostra que o tamanho máximo de lote para essa arquitetura nessa GPU fica em torno de 14 mil imagens, enquanto a **D3ModNetCL** suporta lotes de até 89 mil imagens simultaneamente.

Utilizou-se uma regressão linear na Figura 5.14 de modo a extrapolar a restrição de memória da arquitetura **D3ModNet** e estimar o tempo de inferência para mais de 14 mil imagens. Essa extrapolação permitiu a comparação entre as duas abordagens considerando a capacidade máxima de tamanho de lotes da **D3ModNetCL**. A Figura 5.14 mostra que a abordagem baseada nas **ConformalLayers** é cerca de 1.16 vezes mais rápida que a rede neural convolucional não-associativa.

5.5 Discussão

Ao longo deste Capítulo avaliou-se a técnica proposta como um todo no contexto tanto de qualidade da técnica em termos de acurácia quanto em economia de recursos, ao passo em que se buscou validar algumas das hipóteses construídas ao longo da formulação das ConformalLayers.

No contexto do Experimento I, buscou-se a validação de que a função ReSPro, enquanto função de ativação era, de fato, superior a um modelo totalmente linear definido como baseline. Esse experimento está relacionado diretamente à questão de pesquisa **Q1**, definida da Seção 1.3.1, que busca entender se existe de fato alguma vantagem na utilização da ReSPro como função de ativação quando comparada à uma ativação linear no contexto da acurácia do modelo. Isso foi validado apesar de o modelo ter se mostrado ligeiramente mais limitado quando comparado a uma função de ativação como a ReLU. Tal resultado se deve, sobretudo, à folga em relação ao parâmetro α estimado e a norma L^2 . Um parâmetro mais ajustado levaria a resultados mais similares aos obtidos com a ReLU. Além disso, modelos com uma única camada normalmente não apresentam capacidade de generalização, posto que muitas vezes não conseguem modelar a complexidade da base de dados. Assim, foi desenhado o Experimento II, que visava comparar os resultados numa arquitetura capaz de prover uma resposta mais adequada à complexidade dos dados. Esse experimento, ligado à questão de pesquisa **Q2** (Seção 1.3.1), demonstrou que a ReSPro apresenta capacidade de aprendizado próximo ao dos modelos implementados com ReLU, ainda que a folga no parâmetro α imponha algumas limitações a esse aprendizado.

O Experimento III visava a validação da ideia de tempo de inferência constante e consistia em avaliar o tempo de inferência conforme a profundidade da rede era incrementada. Associado à questão de pesquisa **Q3** (Seção 1.3.1), o experimento mostrou que o comportamento esperado de fato ocorria e que o tempo de inferência se mantém praticamente constante, ainda que um pouco alto. Isso se deve, sobretudo, ao número constante de operações executadas na inferência. O experimento concluiu que, para a arquitetura utilizada, a aplicação das ConformalLayers seria vantajosa para redes mais profundas, quando analisado unicamente o quesito de tempo de inferência. Acredita-se que boa parte do tempo de processamento necessário ao processo de inferência realizado por redes baseadas em ConformalLayers seja proveniente da ineficiência das bibliotecas de matrizes esparsas quando comparadas aos mesmos procedimentos implementados para matrizes densas. O PyTorch, no entanto, é um dos frameworks que tem investido algum tempo na melhoria das operações envolvendo estruturas de dados esparsas, melhorando

gradualmente a performance das implementações disponibilizadas.

O Experimento IV demonstrou uma das maiores vantagens na utilização de abordagens associativas: a economia de recursos. Por reduzir consideravelmente o número de operações e tensores intermediários armazenados, as ConformalLayers acabam por permitir a execução de lotes maiores em menor tempo e consumindo menos memória. Essa característica apresenta-se como uma possível resposta para a questão de pesquisa definida em **Q4** (Seção 1.3.1). Esse tipo de ganho torna esse ferramental bastante apropriado para dispositivos com recursos computacionais limitados, como dispositivos móveis, roteadores inteligentes e dispositivos IoT em geral, os chamados dispositivos de borda.

Capítulo 6

Conclusões e Trabalhos Futuros

Este trabalho apresentou, em linhas gerais, uma abordagem para a construção de redes neurais convolucionais sequenciais de maneira associativa. Ao utilizar transformações simples como reflexão, escala e projeção aplicadas sobre um ponto n -dimensional que representa o conjunto de entradas da rede foi possível, através do mergulho no modelo conforme de geometria, prover uma formulação de função de ativação modelável de maneira linear mas com interpretação linear. Essa função, chamada **ReSPro** e inicialmente desenhada em álgebra geométrica, é uma função de ativação não-linear e diferenciável e pode ser apresentada num formato tensorial e associativo, definido utilizando-se um tensor esparsos de rank-2, *i.e.*, uma matriz, e um tensor esparsos de rank-3.

A representação das demais camadas tradicionalmente utilizadas em redes neurais convolucionais sequenciais num modelo tensorial compatível com a ReSPro, chamado de **ConformalLayers**, permitiu a composição das camadas de maneira associativa, de modo que toda a porção convolucional pode ser definida também utilizando-se um tensor esparsos de rank-2, *i.e.*, uma matriz, e um tensor esparsos de rank-3.

No que diz respeito às questões que guiaram o desenvolvimento desta tese, os experimentos mostraram que a função desenvolvida apresenta acurácia mais alta que funções de ativação puramente lineares e, embora a acurácia da técnica seja ligeiramente menor quando comparada a técnicas não-associativas com funções de ativação não-lineares, ainda assim se mostra competitiva (ver Q1 na Seção 1.3.1). À medida que os experimentos evoluem e passam a considerar arquiteturas capazes de modelar dados mais complexos, essa aparente desvantagem sob a ótica da acurácia decresce, tornando a abordagem proposta ainda mais relevante no cenário de redes neurais convolucionais sequenciais (ver Q2 na Seção 1.3.1). No que diz respeito ao número de operações executadas, sobretudo durante a inferência, com particular impacto no tempo de execução, a técnica também se apresenta

como viável, principalmente em redes neurais mais profundas (ver Q3 na Seção 1.3.1). Por fim, foi demonstrado que, uma vez que o treinamento esteja concluído, é possível armazenar os tensores compostos através da associatividade e utilizá-los no processo de inferência. Essa abordagem levou a uma expressiva redução no consumo de memória na comparação entre o método proposto e o método tradicional (ver Q4 na Seção 1.3.1).

Observa-se, assim, que a utilização das ConformalLayers provê muitas vantagens em termos de recursos computacionais ao custo de uma pequena fração da acurácia, o que torna a arquitetura bastante apropriada para dispositivos com recursos limitados, como dispositivos de borda (*edge devices*).

6.1 Trabalhos Futuros

Ao longo dessa Seção, serão discutidos brevemente as melhorias e caminhos de pesquisa a serem tomados a partir desse trabalho. A expectativa é que essa abordagem associativa analise as arquiteturas de redes neurais convolucionais existentes atualmente sob uma nova ótica e gere um novo leque de trabalhos baseados em arquiteturas associativas.

6.1.1 Ajuste mais Preciso do Parâmetro α

Uma das limitações da técnica diz respeito à acurácia relativamente menor quando comparada a técnicas tradicionais. Como discutido anteriormente, um dos motivos que leva a esse comportamento é a folga existente entre o parâmetro α da ReSPro e a norma L^2 do vetor de entrada de cada camada. Visando mitigar esse problema, uma linha de pesquisa seria propor um melhor ajuste para esse parâmetro. Algumas direções podem ser seguidas aqui, desde o aprendizado automático de um fator detrator para o parâmetro α até a obtenção desse valor através do *ensemble* com outras técnicas, semelhante ao desenvolvido por Tetko [29].

6.1.2 Representação de Camadas Totalmente Conectadas

Atualmente as ConformalLayers não possuem implementação para camadas totalmente conectadas. Futuramente, pretende-se incluir esse tipo de camada como uma ConformalLayer partindo-se do pressuposto de que uma camada totalmente conectada (desconsiderando-se a ativação) corresponde a uma transformação linear aplicada sobre um vetor de entrada. Assim, é possível modelar essa transformação com álgebra geomé-

trica visando sua inclusão no conjunto de módulos disponíveis hoje.

6.1.3 Modelagem do viés em Camadas Convolucionais

Outra limitação que também gera impacto na acurácia diz respeito a ausência de bias no módulo convolucional das ConformalLayers. Uma direção de pesquisa bastante promissora diz respeito à modelagem do viés como uma translação a ser aplicada ao vetor de entrada. Essa translação é também modelável com álgebra geométrica e representável com tensores, restando apenas a verificação de como essa abordagem se encaixaria na expressão final das ConformalLayers.

6.1.4 Aplicação das ConformalLayers no processo de treinamento

Atualmente a grande vantagem das ConformalLayers são exploradas apenas no processo de inferência. Durante o processo de treinamento, as ConformalLayers também apresentam vantagens sob o ponto de vista teórico: ao compor as camadas de modo associativo, menos gradientes seriam calculados, de modo que o processo se tornaria menos custoso. Do ponto de vista prático, no entanto, a atualização da cache entre as várias iterações aumenta o tempo necessário ao treinamento. A exploração de outras metodologias para o treinamento utilizando a abordagem associativa se apresenta como um possível desdobramento desta pesquisa.

6.1.5 ConformalLayers em outras arquiteturas de redes e tarefas de aprendizado de máquina

Nos próximos trabalhos, pretende-se avaliar a viabilidade da utilização das ConformalLayers em outras tarefas de aprendizado de máquina e visão computacional, como segmentação de imagens, por exemplo. Além disso, tendo em mente que a representação de camadas totalmente conectadas pode ser implementada em breve, a utilização da abordagem associativa em MLPs genéricas parece ser o último estágio antes da implementação de outras arquiteturas como *autoencoders* e redes recorrentes, por exemplo.

6.1.6 Análise do desempenho das ConformalLayers em outras arquiteturas de GPU

Arquiteturas mais recentes de GPU, como a Ampere, possuem melhor desempenho quando comparadas à arquitetura Pascal, utilizada nos experimentos descritos nessa tese, sobretudo quando se trata de dados esparsos. Parece existir, assim, uma tendência natural a se explorar melhor o potencial desse tipo de representação. Desse modo, um dos possíveis caminhos decorrentes desse trabalho é uma análise detalhada da performance da implementação apresentadas em GPUs com essa arquitetura.

Referências

- [1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.
- [2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Massachusetts Institute of Technology, 2016.
- [3] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 234–241. Springer, 2015.
- [5] Tangquan Qi, Yong Xu, Yuhui Quan, Yaodong Wang, and Haibin Ling. Image-based action recognition using hint-enhanced deep neural networks. *Neurocomputing*, 267:475–488, 2017.
- [6] Earnest Paul Ijjina and Chalavadi Krishna Mohan. Hybrid deep neural network model for human action recognition. *Applied Soft Computing*, 46:936–952, 2016.
- [7] Leandro A. F. Fernandes, Carlile Lavor, and Manuel M. Oliveira. *Álgebra Geométrica e Aplicações*, volume 85 of *Notas em Matemática Aplicada*. SBMAC, 2017.
- [8] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.
- [9] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.
- [10] Ernst Julius Berg. *Heaviside’s Operational Calculus as applied to Engineering and Physics*. McGraw-Hill Book Company, Inc., 1936.
- [11] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.
- [12] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.
- [13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

- [14] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning*, volume 30, page 3, 2013.
- [15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.
- [16] D. Hendrycks and K. Gimpel. Gaussian error linear units (GELUs). arXiv: 1606.08415, 2016.
- [17] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7, 2017.
- [18] Samta Gupta and Susmita Ghosh Mazumdar. Sobel edge detection algorithm. *International Journal of Computer Science and Management Research*, 2(2):1578–1583, 2013.
- [19] Guang Deng and LW Cahill. An adaptive gaussian filter for noise reduction and edge detection. In *Proceedings of the IEEE Conference Record Nuclear Science Symposium and Medical Imaging Conference*, pages 1615–1619, 1993.
- [20] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020.
- [21] L. Dorst, D. Fontijne, and S. Mann. *Geometric Algebra for Computer Science: an Object-oriented Approach to Geometry*. Elsevier, 2010.
- [22] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- [23] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, 2019.
- [24] Rahul Mehta. Sparse transfer learning via winning lottery tickets. *arXiv preprint arXiv:1905.07785*, 2019.
- [25] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.
- [26] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS’16, page 4114–4122, Red Hook, NY, USA, 2016. Curran Associates Inc.
- [27] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

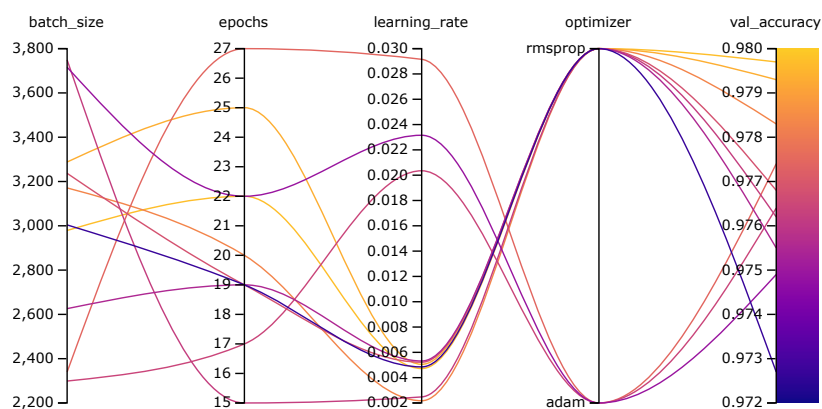
- [28] Hamed Omidvar, Vahideh Akhlaghi, Hao Su, Massimo Franceschetti, and Rajesh Gupta. Associative convolutional layers. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3115–3123. PMLR, 13–15 Apr 2021.
- [29] I. V Tetko. Associative neural network. *Neural Processing Letters*, 16(2):187–199, 2002.
- [30] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.
- [31] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.
- [32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.
- [33] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [34] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer. Squeezenet: unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 129–137, 2017.
- [35] O. Saha, A. Kusupati, H. V. Simhadri, M. Varma, and P. Jain. RNNPool: efficient non-linear pooling for RAM constrained inference. In *Neural Information Processing Systems*, pages 20473–20484, 2020.
- [36] M. Alwani, H. Chen, M. Ferdman, and P. Milder. Fused-layer CNN accelerators. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, 2016.
- [37] M. Tan and Q. Le. Efficientnet: rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 6105–6114, 2019.
- [38] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: efficient convolutional neural networks for mobile vision applications. *arXiv: 1704.04861*, 2017.
- [39] J.K. Pearson and D.L. Bisset. Neural networks in the clifford domain. In *Proceedings of IEEE International Conference on Neural Networks*, volume 3, pages 1465–1469 vol.3, 1994.
- [40] Sven Buchholz. *A theory of neural computation with Clifford algebras*. PhD thesis, Christian-Albrechts Universität Kiel, 2005.

- [41] Sven Buchholz and Gerald Sommer. On clifford neurons and clifford multi-layer perceptrons. *Neural Networks*, 21(7):925–935, 2008.
- [42] Juan Pablo Serrano Rubio, Arturo Hernández Aguirre, and Rafael Herrera Guzmán. A conic higher order neuron based on geometric algebra and its implementation. In *Mexican International Conference on Artificial Intelligence*, pages 223–235. Springer, 2012.
- [43] Kamran Chitsaz, Mohsen Hajabdollahi, Nader Karimi, Shadrokh Samavi, and Shahraram Shirani. Acceleration of convolutional neural network using fft-based split convolutions. *CoRR*, abs/2003.12621, 2020.
- [44] David Hestenes. *New foundations for classical mechanics*, volume 15. Springer Science & Business Media, 2012.
- [45] Kenneth Hoffman and Ray A. Kunze. *Linear Algebra*. PHI Learning, 2004.
- [46] R. Larson. *Precalculus: A Concise Course*. Cengage Learning, 2010.
- [47] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Pearson, 3 edition, 2008.
- [48] Vladimir Kazeev, B. Khoromskij, and E. Tyrtysnikov. Multilevel toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity. *Society for Industrial and Applied Mathematics Journal on Scientific Computing*, 35, 05 2013.
- [49] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.
- [50] W. H. Young. On the multiplication of successions of Fourier constants. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 87(596):331–339, 1912.
- [51] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.
- [52] Y. LeCun, C. Cortes, and C. J. Burges. MNIST handwritten digit database. ATT Labs, 2010.
- [53] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv: 1708.07747, 2017.
- [54] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.
- [55] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.
- [56] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: a novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

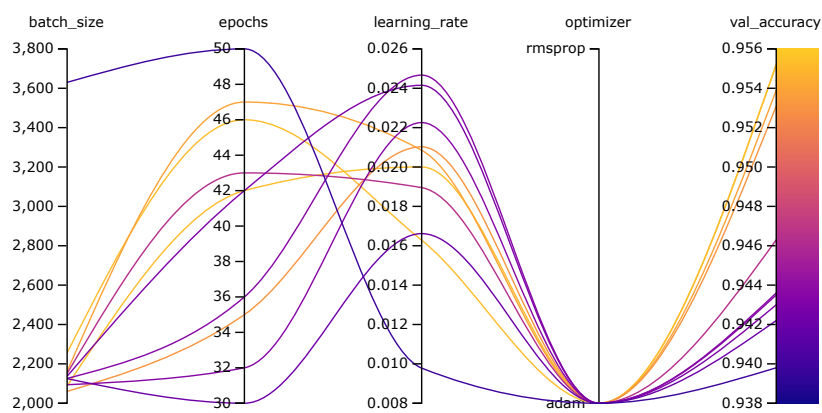
-
- [57] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
 - [58] Hartwig Anzt, Yuhsiang M. Tsai, Ahmad Abdelfattah, Terry Cojean, and Jack Dongarra. Evaluating the performance of nvidia’s a100 ampere gpu for sparse and batched computations. In *IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 26–38, 2020.

APÊNDICE A – Seleção de Hiperparâmetros

Este Apêndice apresenta alguns dos resultados da seleção de hiperparâmetros aplicada sobre as rede baseadas na arquitetura LeNet-5, discutida no Capítulo 5. Os hiperparâmetros foram selecionados utilizando-se uma varredura pelo espaço de hiperparâmetros guiada por uma abordagem Bayesiana implementada na ferramenta *Weights and Biases* [55]. As Figuras A.1, A.2, e A.3 apresentam os gráficos indicando os hiper parâmetros selecionados para as top-10 acurácias de validação obtidos pelas redes **LeNet** e **LeNetCL** durante o treinamento utilizando as bases de dados MNIST [52], FashionMNIST [53], e CIFAR10 [54], respectivamente. Após essa seleção inicial, cada par rede neural/base de dados foi treinado durante 200 épocas.

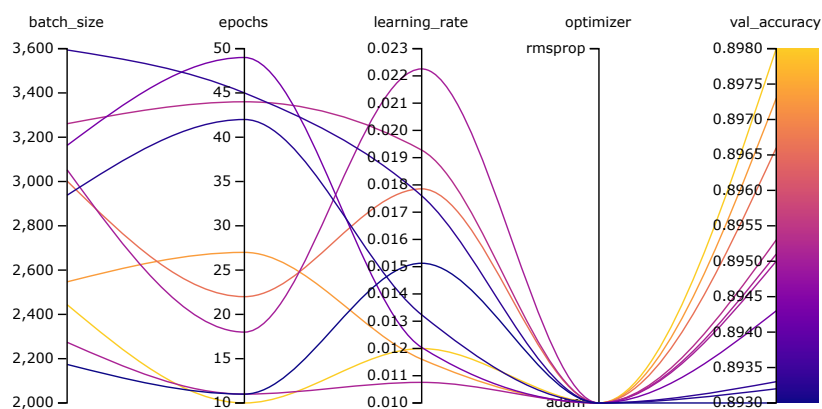


(a) LeNet

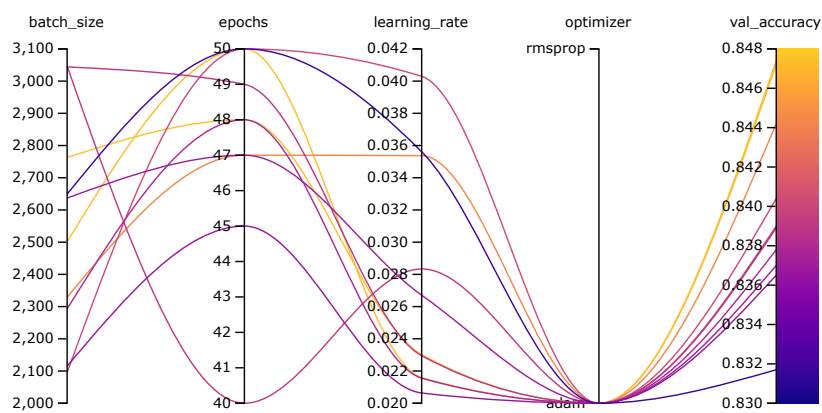


(b) LeNetCL

Figura A.1: Hiperparâmetros selecionados para as redes neurais baseadas na LeNet-5 [57] que levaram às top-10 acurácias de validação utilizadas no Experimento II na base de dados MNIST.

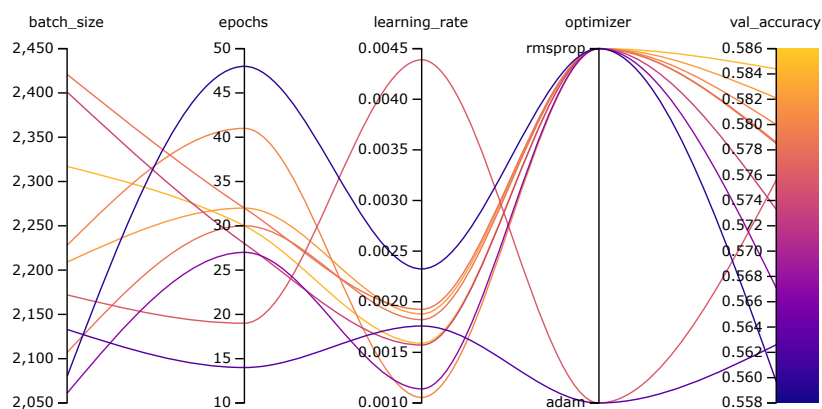


(a) LeNet

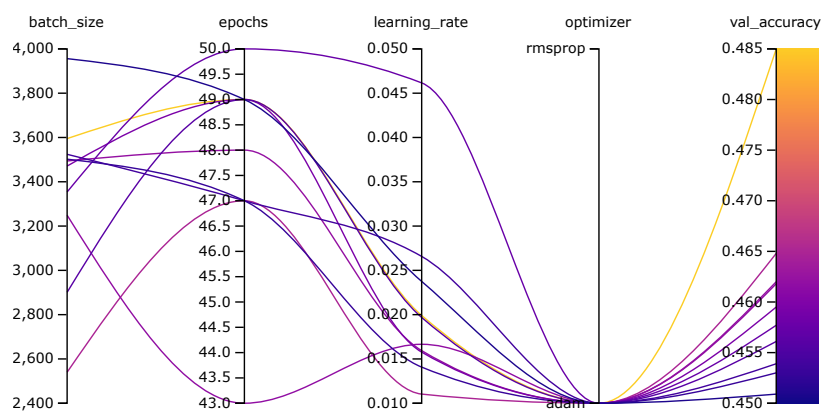


(b) LeNetCL

Figura A.2: Hiperparâmetros selecionados para as redes neurais baseadas na LeNet-5 [57] que levaram às top-10 acurácias de validação utilizadas no Experimento II na base de dados FashionMNIST.



(a) LeNet



(b) LeNetCL

Figura A.3: Hiperparâmetros selecionados para as redes neurais baseadas na LeNet-5 [57] que levaram às top-10 acurácias de validação utilizadas no Experimento II na base de dados CIFAR10.