GABRIEL DE SOUZA BARRETO

UM ESTUDO SOBRE *NEURAL ORDINARY DIFFERENTIAL EQUATIONS*

(*versão pré-defesa, compilada em 11 de dezembro de 2019*)

Trabalho de Graduação apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná..

Área de concentração: *Ciência da Computação*.

Orientador: David Menotti.

CURITIBA PR

2019

RESUMO

A partir da introdução do *Multi-Layer Perceptron*, as redes neuronais artificiais foram extensivamente estudadas, resultando no desenvolvimento de diversos modelos com arquiteturas próprias. A proposta de um modelo de aprendizado residual profundo, a rede neuronal ResNet, proporcionou uma solução que pode se beneficiar do uso de um grande número de camadas sem sofrer do problema de degradação encontrado nos outros modelos convencionais. Dessa forma, é possível a criação de redes neuronais extremamente profundas, permitindo assim a interpretação da dinâmica das camadas como uma aproximação de um sistema com transformações contínuas da entrada. Esse cenário permite interpretar as ResNets como uma solução numérica aproximada para uma determinada equação diferencial ordinária.

Esse novo paradigma motivou o desenvolvimento da *Neural Ordinary Differential Equations*, uma nova família de redes neuronais profundas que parametriza a variação dentre as camadas como uma equação diferencial ordinária. Assim, a saída é calculada utilizando um resolvedor de ODEs que utiliza como parâmetro a profundidade da camada de saída. Essa arquitetura de profundidade contínua permite custo de memória constante, adaptação da estratégia da computação para cada entrada e controle do erro para equilíbrio de precisão numérica com velocidade de processamento. No entanto, esse modelo sofre de limitações e não é capaz de representar qualquer função. Além disso, é necessário o uso do *adjoint sensitivity method* para executar a retropropagação através das camadas do modelo durante o treinamento sem a necessidade de armazenar ou computar novamente as avaliações feitas pelo resolvedor de ODEs (Equações Diferenciais Ordinárias). Os resultados obtidos na utilização do modelo *Neural Ordinary Differential Equations* mostram uma taxa de erros próxima ao encontrado nas ResNets e um número menor de parâmetros que são utizados de forma mais eficiente.

Palavras-chave: Rede Neuronal Profunda, Rede Neuronal Residual, *Neural Ordinary Differential Equations*, Equações Diferenciais Ordinárias, *Adjoint Sensitivity Method*.

ABSTRACT

Since the introduction of a Multi-Layer Perceptron, artificial neural networks were extensively studied, resulting in the development of several models with specific architectures. The proposal of a deep residual learning model, the ResNet neural network, produced a solution that benefits from the use of a large number of layers without suffering from the degradation problem encountered in other conventional models. This way, it is possible to create extremely deep neural networks, allowing an interpretation of the layer dynamics as an approximation of a continuous transformations system of the input. This scenario allows viewing ResNets as an approximated numerical solution for some ordinary differential equation.

This new paradigm motivated the development of Neural Ordinary Differential Equations, a new family of deep neural networks models that parameterizes the variation between layers as an Ordinary Differential Equation. The output is calculated using an ODE Solver that receives the depth of the output layer as a parameter. That continuous depth architecture provides constant memory cost, strategy adaptation for each input evaluation and error control that allows a trade-off between numerical precision and processing speed. However, this model suffers from limitations and is not capable of representing all functions. In addition, it is required using the adjoint sensitivity method to perform backpropagation through all the model's layers during training, so storing values or repeating the ODE Solver's evaluations is not needed. The results found using the Neural Ordinary Differential Equations show an error rate similar to ResNets and a smaller number of parameters that are used more efficiently.

Keywords: Deep Neural Network, Residual Neural Network, Neural Ordinary Differential Equations, Ordinary Differential Equations, Adjoint Sensitivity Method.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

CNN	Rede Neuronal Convolucional
ResNet	Rede Neuronal Residual
UFPR	Universidade Federal do Paraná
DINF	Departamento de Informática
BCC	Bacharelado em Ciência da Computação
CIFAR-10	Base de dados de imagens do Canadian Institute For Advanced Research
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Units
Iter	Número de iterações
ImageNet	Base de dados de imagens organizado em relação com a WordNet da Universidade de Princeton
ODE	Equação Diferencial Ordinária
PVI	Problema de Valor Inicial
NODE	Rede Neuronal de Equações Diferenciais Ordinárias
NFE	Number of Function Evaluations
ILSVRC	mageNet Large Scale Visual Recognition Challenge

LISTA DE SÍMBOLOS

Σ	sigma maiúsculo, representa o somatório
σ	sigma minúsculo, função de ativação
∇	nabla, símbolo do gradiente
$\mathcal{I}$	função identidade
Δ	delta maiúsculo, representa um intervalo
$\mathbb{R}$	conjunto dos números reais
ϕ	phi minúsculo, representa função ou mapeamento
∂	derivada parcial
$M^\intercal$	matriz transposta de M

SUMÁRIO

1 INTRODUÇÃO

Os modelos de aprendizado aprofundado, baseados nas redes neuronais, são bastante utilizados na área de aprendizado de máquina e apresentam várias possibilidades de aplicação. Como apresentado em Hochuli (2019), um neurônio artificial (*perceptron*) é definido com seus respectivos pesos e viés, além de uma função de ativação que determina se o neurônio será ativado a partir de uma certa entrada. Essa estrutura funciona como um classificador binário linear, já que o neurônio só pode estar ativado ou não.

A partir disso, é definida uma rede neuronal artificial (Multi-Layer Perceptron - MLP) que distribui um determinado número de neurônios artificiais de forma interconectada criando uma rede com camadas. Dentre as camadas de neurônios artificiais, é propagado um fluxo que é transmitido pelas conexões até a camada final, onde é obtido o valor resultante para entrada. Isso pode ser observado na Figura 1.1.

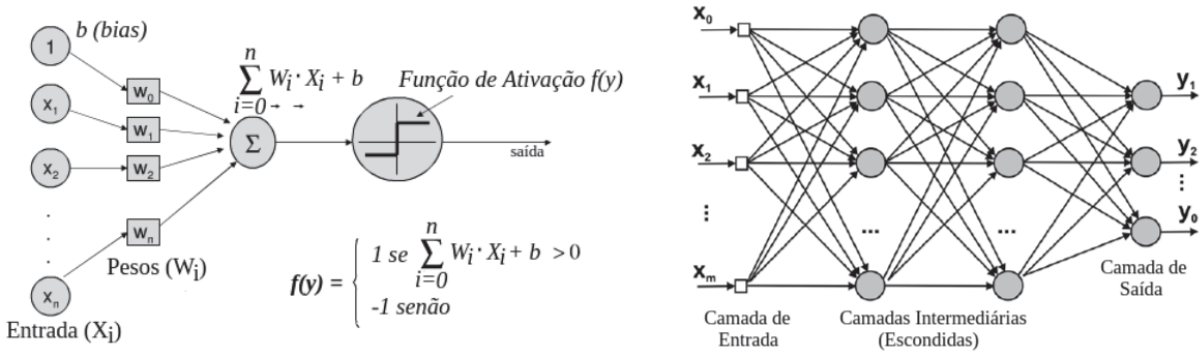

Figura 1.1: Um diagrama apresentando um Neurônio Artificial ou Perceptron (à esquerda) e uma Rede Neuronal Artificial (MLP) com suas camadas de neurônios artificiais (à direita). Imagem de Hochuli (2019).

Dessa forma, as camadas intermediárias (ou camadas escondidas), realizam um mapeamento de um vetor de entrada para um vetor de saída a partir da dinâmica de ativação dos neurônios artificiais do modelo. Para cada saída da rede, existe um erro associado que pode ser calculado com uma função de perda (*Loss*), ao calcular a diferença entre os valores obtidos e os esperados. Para o funcionamento adequado, essa rede neuronal precisa ser treinada com a atualização dos parâmetros de cada neurônio artificial e suas respectivas conexões, a partir de um algoritmo de retropropagação. A retropropagação ocorre com uma transmissão até a camada inicial dos ajustes obtidos na camada de saída a partir do cálculo da função de perda (*Loss*). Após a definição desse modelo básico, várias adaptações possíveis na arquitetura das redes neuronais foram estudadas para a obtenção de melhores resultados na minimização da função de perda.

A profundidade dos modelos, a existência de camadas intermediárias com objetivos específicos ou a presença de conexões entre certas camadas dentro da rede são adaptações que afetam drasticamente os resultados obtidos para cada tipo de problema e sua respectiva base de dados. Isso pode ser ilustrado pelos resultados obtidos em Canziani et al. (2016) ao analisar o

desempenho de modelos diferentes de redes neuronais no desafio ImageNet. As diferenças de acurácia, em problemas de classificação, obtida por alguns modelos podem ser vistas no gráfico da Figura 1.2.

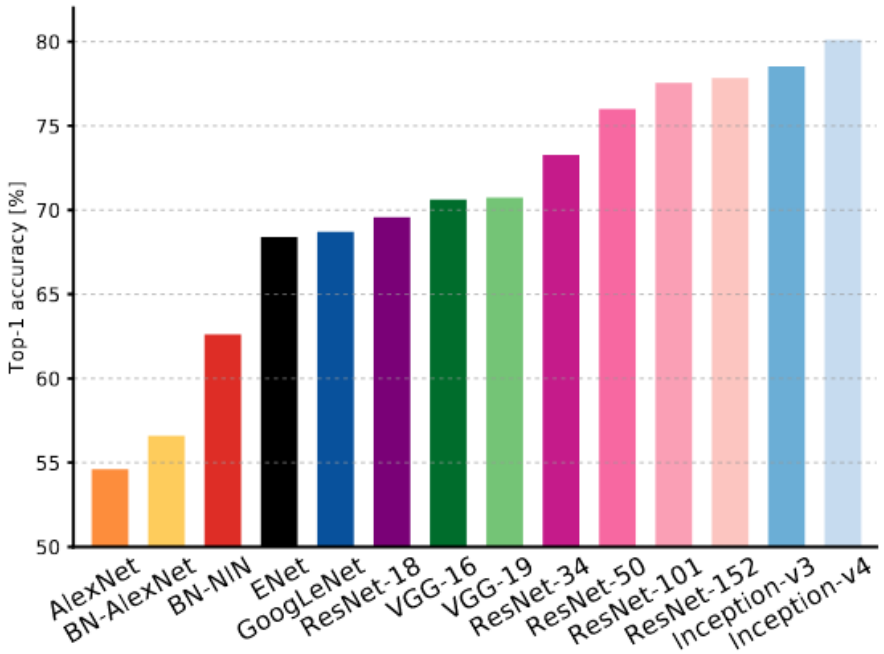

Figura 1.2: Um diagrama dos resultados obtidos por diversos modelos de redes neuronais em relação à acurácia de validação top-1 no desafio ImageNet. Imagem de Canziani et al. (2016).

Esses modelos de redes neuronais são resultantes de estudos que buscam melhorar os resultados em relação à acurácia, ao tempo de treinamento, à eficiência dos parâmetros, ao uso de memória e a muitos outros aspectos relacionados ao desempenho da rede neuronal. Esse estudos têm originado modelos de crescente complexidade e com enorme potencial, capazes de ultrapassar obstáculos que impediam o uso de redes neuronais de formas específicas. Por exemplo, o modelo ResNet apresentado em He et al. (2015) permitiu um grande aumento na profundidade das redes, que era inviabilizado anteriormente pelo fenômeno do desaparecimento e da explosão do gradiente ocorrido em redes com muitas camadas. Esse aumento no número possível de camadas passou a ser tão grande que motivou uma nova interpretação para a dinâmica da propagação na rede, que permite considerar as redes neuronais como uma discretização de sistemas contínuos. A partir disso, foi possível o surgimento da rede Neural Ordinary Differential Equations, um modelo desenvolvido por Chen et al. (2018) que é capaz de transformar a forma que as redes neuronais são interpretadas.

1.1 MOTIVAÇÃO

A definição do modelo de *Neural Ordinary Differential Equations* propicia de imediato uma série de benefícios na sua utilização. Em Chen et al. (2018) são apresentados alguns deles:

- Eficiência no uso da memória: É apresentada na Seção 3.2 a forma de computar os gradientes da função de perda, necessária para o treinamento da rede, utilizando o resolvedor de ODEs e sem o requisito de armazenamento de qualquer valor intermediário da propagação. Isso viabiliza efetuar o treino do modelo com custo constante de memória em relação à profundidade da rede neuronal, o que elimina um gargalo que limita o treinamento de modelos com muitas camadas.

- Computação adaptável: Desde o método de Euler, já se passaram mais de 120 anos de desenvolvimento de resolvedores de ODEs mais eficientes e precisos, como pode ser exemplificado em Runge (1895), Kutta (1901) e Hairer et al. (1993). Além disso, resolvedores modernos propiciam garantias, durante a execução, para alcançar uma determinada tolerância a erro. Essas garantias podem ser em relação ao crescimento do erro de aproximação, ao monitoramento da magnitude do erro e à adaptação da estratégia de avaliação das funções. Isso implica que o custo de computação do modelo possa ser proporcional à complexidade do problema, executando mais computações em casos mais difíceis. No entanto, é possível realizar uma redução da acurácia exigida pelo resolvedor, após o treinamento da rede, permitindo uma execução mais adequada para aplicações de tempo real ou que necessitem de baixo consumo de energia.

- Eficiência dos parâmetros: Quando a dinâmica das camadas escondidas é parametrizada como uma função contínua, os parâmetros das camadas que são alterados durante o treinamento são fixados. Isso reduz o número de parâmetros requeridos para a execução de treinamento supervisionado, independentemente do número de computações feitas pelo resolvedor de ODEs, como é apresentado na Seção 3.3.

1.2 OBJETIVOS

A Rede Neuronal de Equações Diferenciais Ordinárias (NODE) é um modelo que é introduzido como uma adaptação das ResNets para uma dinâmica contínua, baseando o seu funcionamento no uso de resolvedores de ODEs como uma caixa preta para a computação dos valores de saída em qualquer camada. O primeiro objetivo deste trabalho é o entendimento do seu funcionamento, seguido de posterior investigação sobre a validade dessa proposta a partir do ponto de vista teórico. Para melhor embasamento, é pertinente um amplo estudo dos trabalhos relacionados e modelos propostos que antecederam e inspiraram a NODE. Em seguida, será feita uma análise dos resultados apresentados na proposição do modelo, verificando o seu embasamento e interpretando o comportamento apresentado. Após esse entendimento, segue o objetivo de estudar as eventuais limitações da NODE, de forma a delimitar um escopo da sua utilização e traçar os paralelos entre o seu desempenho e as ResNets em casos específicos. Finalmente, os experimentos feitos buscam averiguar em outras bases a validade da teoria apresentada e comparar os resultados encontrados.

1.3 ESTRUTURA DO DOCUMENTO

Além da descrição introdutória apresentada no Capítulo 1, incluindo a motivação e os objetivos do trabalho desenvolvido, este documento é estruturado com mais quatro capítulos. No Capítulo 2, é apresentada uma revisão literária que trata do modelo de aprendizado residual profundo, fazendo uma descrição do cenário anterior às ResNets, o detalhamento desse modelo e a apresentação de estudos interpretando as redes neuronais como um sistema dinâmico contínuo. No capítulo 3, o modelo NODE (*Neural Ordinary Differential Equations*) é detalhadamente explicado, incluindo a sua arquitetura, a sua forma de treinamento, os seus resultados e as suas limitações. Os experimentos são relatados no Capítulo 4, apresentando a arquitetura dos modelos, as bases de dados utilizadas e os resultados encontrados. Finalmente, são feitas algumas considerações finais e é apresentada a conclusão do trabalho no Capítulo 5.

2 REVISÃO LITERÁRIA

Antes de descrever o modelo Neural Ordinary Differential Equations (Rede Neuronal de Equações Diferenciais Ordinárias), é interessante discorrer brevemente sobre o modelo de aprendizado residual profundo (ResNet), além das abordagens de redes neuronais como sistemas dinâmicos contínuos.

2.1 REDES PRÉ RESNET

As pesquisas relacionadas aos modelos de redes neuronais têm obtido crescentemente melhores resultados ao solucionar problemas específicos. As CNNs, por exemplo, permitiram avanços na classificação de imagens, como apresentado em Krizhevsky et al. (2017). Além das possibilidades de modelos específicos para problemas diferentes, as redes neuronais podem facilmente integrar características de alto, médio e baixo nível com classificadores em múltiplas camadas sequenciais, sendo que o nível dessas características pode ser enriquecido ao aumentar o número de camadas (profundidade) da rede, conforme afirmado em He et al. (2015). Isso é confirmado por pesquisas recentes e, como é exemplificado em Simonyan e Zisserman (2014), a profundidade da rede neuronal é extremamente importante para o comportamento do modelo.

No entanto, o desempenho das redes neuronais não melhora sempre com o aumento do número de camadas, podendo até piorar em muitos casos. O principal causador desse problema é o fenômeno de desaparecimento e explosão do gradiente, como já tinha sido apresentado em Bengio et al. (1994) e novamente discutido em Glorot e Bengio (2010). Mesmo com o uso de soluções paliativas, como efetuar uma normalização inicial e utilizar camadas intermediárias de normalização, as redes mais profundas começam a convergir, mas estão sujeitas a um problema de degradação, como é apresentado em He et al. (2015). Esse problema de degradação não ocorre por sobreajuste e implica que acrescentar mais camadas na rede neuronal só causa maior erro de treinamento, conforme exemplificado na Figura 2.1.

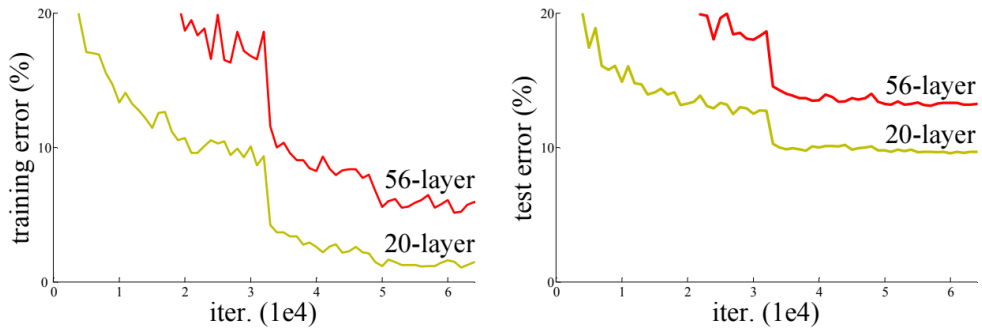

Figura 2.1: Os erros de treinamento (à esquerda) e de teste (à direita) na CIFAR-10 com redes neuronais simples (convencionais) com profundidades de 20 e 56 camadas. A rede mais profunda apresenta erros maiores nos dois casos. Imagem de He et al. (2015).

2.2 RESNET

O problema da degradação é analisado em He et al. (2015), sendo a motivação para a introdução de um modelo de aprendizado residual profundo, a rede neuronal ResNet, com o objetivo de resolver esse problema. Sendo x a entrada inicial, $\mathcal{H}(x)$ é a função que determina um bloco de camadas e $\mathcal{F}(x)$ é definida como uma função residual tal que $\mathcal{F}(x) := \mathcal{H}(x) - x$. Ao assumir que a saída de $\mathcal{F}(x)$ é da mesma dimensão que x, é possível afirmar que $\mathcal{F}(x) + x$ é aproximadamente $\mathcal{H}(x)$, de forma que um bloco cuja operação se aproxime de $\mathcal{F}(x)$ e acrescente x ao resultado é equivalente ao bloco $\mathcal{H}(x)$ inicial.

Como é apresentado na Figura 2.2, a formulação de $\mathcal{F}(x) + x$ pode ser interpretada como uma rede neuronal com conexões de atalho. Essas conexões são uma função identidade, que apenas retornam a entrada do bloco e viabilizam somar cada entrada ao seu respectivo resultado da operação do bloco. Essa formulação permite a criação de um modelo de rede neuronal que não possui nenhum parâmetro adicional ou maior complexidade computacional, além de poder ser treinada com *Stochastic Gradient Descent* (SGD) utilizando a retropropagação, como apresentado em He et al. (2015).

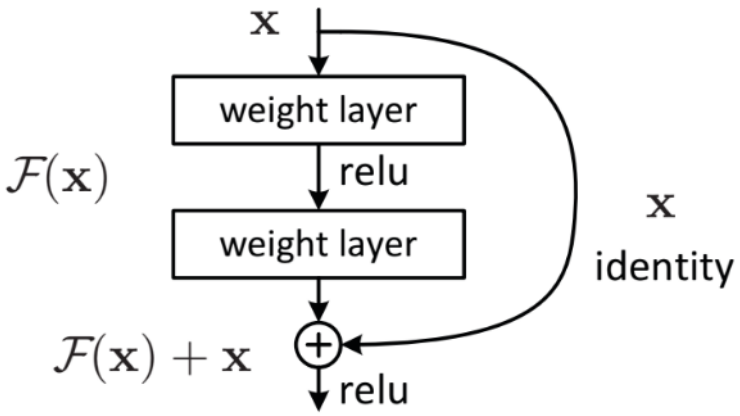

Figura 2.2: Um bloco do modelo ResNet. Imagem de He et al. (2015).

A ResNet apresenta essa formulação alternativa, utilizando um condicionamento prévio para lidar com o problema da degradação. Caso a função ótima a ser aproximada pelo bloco for próxima a uma função identidade, é mais fácil o bloco mapear as pequenas perturbações na entrada do que aprender uma nova função, como é afirmado em He et al. (2015). A Figura 2.3 mostra como as saídas das ResNets apresentam menores variações quando comparadas às redes convencionais (chamadas de *plain* nos gráficos).

A análise da Figura 2.3 revela que as camadas das ResNets geralmente apresentam respostas de menor intensidade que as alternativas convencionais. Esses resultados atendem à motivação inicial das ResNets, na forma que os blocos residuais executariam perturbações menores na entrada quando comparados com comportamento das camadas de redes convencionais. Além disso, é possível perceber a partir dos gráficos que as alterações realizadas por cada camada sobre a sua entrada são mais amenas a medida que as ResNets se tornam mais profundas. Cada

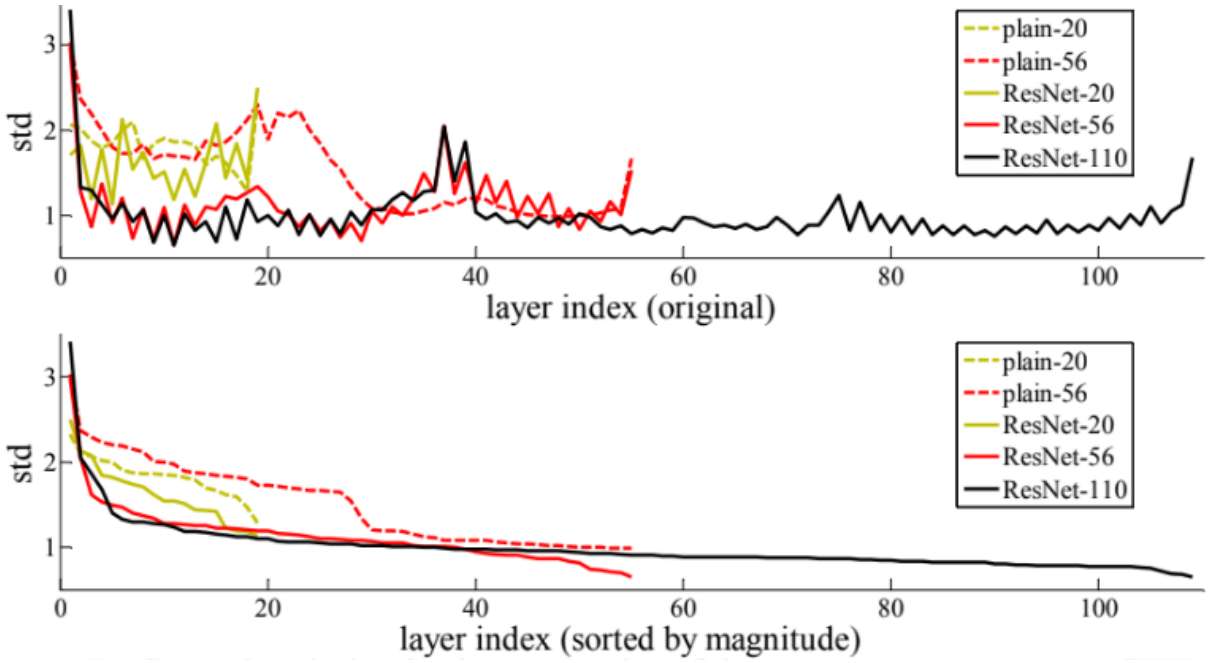

Figura 2.3: Gráficos relacionados aos experimentos feitos na CIFAR-10. O eixo vertical apresenta o desvio padrão (std) nas saídas de cada camada em cada um dos modelos de rede neuronal utilizados, demonstrando a variação da saída em relação à entrada em cada caso. Foram consideradas as saídas de cada camada 3x3 após *Batch Normalization* e antes da aplicação da função de ativação não linear (ReLU/adição). No topo as camadas são apresentadas na ordem original e embaixo as saídas são apresentadas em ordem decrescente. Imagem de He et al. (2015).

bloco residual, individualmente, tende a modificar menos a entrada nos casos de ResNets que contém mais camadas.

As conexões de atalho são usadas a cada determinado grupo de camadas, criando os blocos de aprendizado residual na forma já apresentada na Figura 2.2. Dado x o vetor de entrada e y o vetor de saída, sendo $\mathcal{W}_i$ o conjuto dos pesos de um determinado bloco, é definido como

$$y = \mathcal{F}(x, \{W_i\}) + x \tag{2.1}$$

A função $\mathcal{F}(x, \{W_i\})$ representa o mapeamento residual a ser aprendido pelo bloco. Por exemplo, na Figura 2.2 o bloco apresenta apenas duas camadas. Então $\mathcal{F} = W_2\sigma(W_1x)$, onde σ denota ReLU como visto em Nair e Hinton (2010), sendo o viés omitido para simplificar a notação. A aplicação da segunda ativação ReLU acontece apenas após a operação $\mathcal{F} + x$. É importante ressaltar que as dimensões de x e $\mathcal{F}$ devem ser iguais. Quando isso não ocorre, é necessário aplicar uma projeção linear $\mathcal{W}_s$ nas conexões de atalho do bloco residual para igualar as dimensões de forma que

$$y = \mathcal{F}(x, \{W_i\}) + W_sx \tag{2.2}$$

A adição das conexões de atalho é conveniente já que, enquanto não introduz complexidade computacional ou parâmetros adicionais, permite uma comparação direta com redes neuronais convencionais. Alguns exemplos desses resultados são apresentados em He et al.

(2015), sendo as arquiteturas utilizadas nos experimentos descritas na Figura 2.4 e os gráficos dos erros encontrados em cada caso apresentados na Figura 2.5:

Analisando os resultados das redes convencionais, é percebido que o erro de validação é mais alto em 34 do que em 18 camadas. Além disso, mesmo com o espaço da solução com profundidade 18 sendo um subespaço do caso de 34 camadas, a rede mais profunda apresenta um erro maior durante todo o treinamento do modelo.

Já no caso das ResNets, a análise dos resultados demonstra que a ResNet mais profunda obtém percentual de erro consideravelmente mais baixo que a rede residual de 18 camadas. Ainda mais importante, a ResNet de 34 camadas apresenta erro menor durante o treinamento e melhor generalização para os dados de validação. Isso indica que o problema de degradação foi solucionado nesse modelo, permitindo um ganho de acurácia a partir de aumento de profundidade na ResNet. Finalmente, é possível constatar que a ResNet com 18 camadas obtém erro semelhante ao apresentado na rede convencional correspondente, não alcançando um ganho de acurácia por ter pouca profundidade. No entanto, o modelo ResNet converge mais rápido (como pode ser destacado no comportamento até iter. 15 1e4 dos gráficos na Figura 2.5 por exemplo), facilitando a otimização em estágios iniciais e demonstrando execução mais eficiente de SGD mesmo em uma rede pouco profunda.

2.3 REDES NEURONAIS CONTÍNUAS

Com a possibilidade de redes neuronais mais profundas que não sofrem com o problema de degradação e apresentam resultados superiores, algumas pesquisas investigaram a ideia da introdução de ainda mais camadas nas redes. As redes neuronais profundas podem ser interpretadas como uma discretização de sistemas dinâmicos contínuos, como afirmado em Weinan (2017). Neste sistema dinâmico contínuo, cada passo é uma transformação linear seguida por uma função de ativação não linear aplicada a cada componente.

As ResNets também podem ser interpretadas dessa forma, sendo então descritas como o seguinte sistema dinâmico discreto: Dado que z_l e z_{l+1} são, respectivamente, a entrada e a saída da l-ésima camada da ResNet, além de dado um elemento y_l sendo uma variável auxiliar da camada l (He et al., 2016)

$$y_l = h(z_l) + \mathcal{F}(z_l, W_l) \tag{2.3}$$

$$z_{l+1} = g(y_l) \tag{2.4}$$

Tanto a função h relacionada a possíveis transformações nas conexões de atalho quanto a função g relacionada a quaisquer alterações nas ativações podem ser não lineares. Após vários experimentos numéricos, (He et al., 2016) apresenta um cenário no qual o treinamento é mais

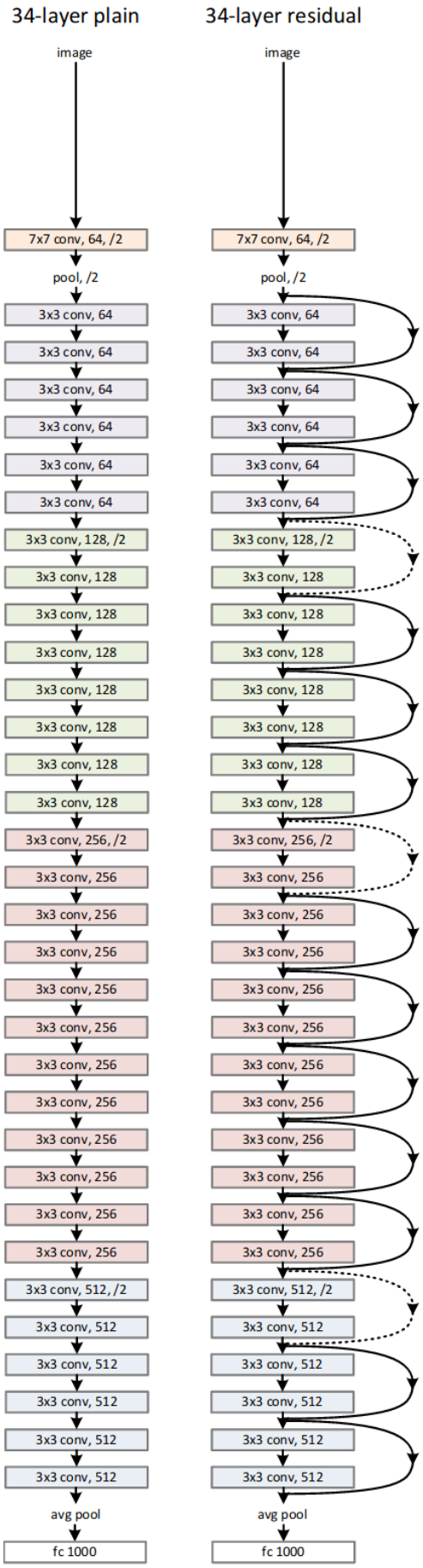

Figura 2.4: Diagrama demonstrando a arquitetura das redes neuronais utilizadas nos experimentos de He e Zhang (2015). Uma rede convencional de 34 camadas é apresentada à esquerda. A ResNet correspondente está à direita, com conexões de atalho a cada par de filtros 3x3. Essas conexões são representadas pelas setas à direita, sendo que a linha pontilhada representa os casos onde é necessária a realização de uma transformação linear da entrada. As arquiteturas utilizadas para cada camada são definidas de forma similar para ambos modelos. Imagem de He et al. (2015).

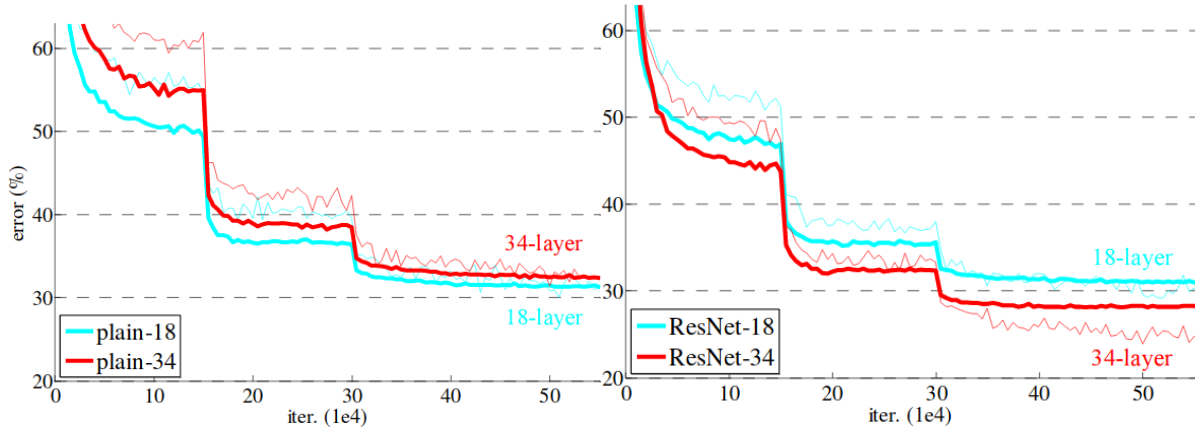

Figura 2.5: Treinamento e validação das redes neuronais na execução de central crops na base ImageNet. O gráfico à esquerda apresenta os resultados das redes neuronais convencionais, de 18 e 34 camadas, enquanto são descritos à direita os resultados das ResNets correspondentes. As curvas de traços finos apresentam os erros de treinamento, enquanto as linhas em negrito representam os erros de validação do respectivo modelo. As ResNets apresentadas não utilizam nenhum parâmetro extra em comparação com as redes neuronais convencionais de profundidade respectiva. Imagem de He et al. (2015).

fácil em redes neuronais muito profundas (milhares ou centenas de camadas) se tanto g quanto h forem funções identidade.

Conforme afirmado em Weinan (2017)), esse resultado é o que seria esperado a partir do ponto de vista de sistemas dinâmicos e, ao denotar $\mathcal{G}$ como a função inversa de g, o sistema dinâmico pode ser reescrito como

$$z_{l+1} = \mathcal{G}(h(z_l) + \mathcal{F}(z_l, W_l)) \tag{2.5}$$

Para evitar o fenômeno de desaparecimento e explosão do gradiente, o gradiente da parte à direita da Equação 2.5 deve ser aproximar a uma função identidade $\mathcal{I}$. Assumindo que $\mathcal{F}$ resulta em uma pequena variação, isso é então satisfeito por

$$\nabla\mathcal{G}\nabla h \sim \mathcal{I} \tag{2.6}$$

Essa condição já é atendida quando g e h são funções identidade. De forma geral, é definido então

$$z_{l+1} \sim \mathcal{G}(h(z_l)) + \nabla\mathcal{G} \cdot \mathcal{F}(z_l, W_l) \tag{2.7}$$

Nessa notação, é simples observar que o termo de maior ordem de magnitude é dominado por $\mathcal{G}(h)$. Dessa forma, como é destacado em Weinan (2017), a flexibilidade obtida por generalizar a escolha das funções g e h não traz uma melhora considerável na utilização de

um grande número de camadas. Sendo assim, dadas g e h funções identidade, a Equação 2.5 pode ser reformulada como

$$z_{l+1} = z_l + \mathcal{F}(z_l, W_l) \tag{2.8}$$

No entanto, a Equação 2.8 pode ser considerada como sendo uma discretização do sistema dinâmico

$$\frac{dz}{dl} = \mathcal{F}(z, W(l)) \tag{2.9}$$

Sendo assim, a discretização mais simples da Equação 2.9 é definida como

$$z_{l+1} = z_l + \Delta t_l \mathcal{F}(z_l, W_l) \tag{2.10}$$

onde t_l é definido como o tamanho do l-ésimo passo e, conforme em Lambert (1991), tanto a eficiência quanto a estabilidade do algoritmo podem ser melhoradas simultaneamente ao adaptar corretamente o t_l escolhido. Em redes neuronais, essa adaptação ocorre na escolha do número de camadas. Essa análise e reformulação permitem definir então uma relação direta entre o modelo de redes neuronais profundas e a discretização de ODEs, inclusive as ResNets, como apresentado em Lu et al. (2017).

3 NEURAL ORDINARY DIFFERENTIAL EQUATIONS

A partir dos bons resultados obtidos pelas ResNets e das possibilidades oferecidas pela interpretação das camadas de redes neuronais como uma discretização de um sistema contínuo, relacionar o comportamento desses modelos com Equações Diferenciais Ordinárias (ODEs) se mostra um caminho bastante promissor. Com alguns ajustes na arquitetura e no treinamento do modelo, essa relação pode ser ainda mais próxima.

3.1 NEURAL ODE

Como foi mostrado anteriormente na Equação 2.8, a saída z_{l+1} da camada l de uma determinada ResNet pode ser definida pela sua entrada z_l somada ao resultado da aplicação da função aproximada por aquela camada $\mathcal{F}(z_l, W_l)$. Sendo T a última camada da ResNet (profundidade máxima) e *0* a primeira camada dessa rede neuronal, esse mapeamento definido é válido para todo $l \in \{0..T\}$, dado que a entrada z_l e $\mathcal{F}(z_l, W_l)$ tenham sempre a mesma dimensão. Conforme apresentado em Lu et al. (2017), essas atualizações iterativas a cada camada podem ser consideradas como uma discretização de uma equação diferencial através do uso do Método de Euler. Isso foi analisado com mais profundidade em Haber e Ruthotto (2017), além de ter sido feita uma demonstração mais detalhada dessa relação. De fato, uma ResNet com muitas camadas e incrementos bastante pequenos, apresenta uma dinâmica aproximadamente contínua. Essa ResNet pode ser parametrizada por uma ODE como apresentado em Chen et al. (2018):

$$\frac{dz(l)}{dl} = \mathcal{F}(z(l), l, W(l)) \tag{3.1}$$

É definido assim o modelo da Rede Neuronal de Equações Diferenciais Ordinárias (NODE). Considerando os parâmetros *W(l)* constantes para todas camadas da rede neuronal, a função $\mathcal{F}$ depende apenas de *z(l)* e *l*. Como foi detalhado em Dupont et al. (2019), basta então formular a transformação a partir de uma camada l para outra *l+1* em uma ResNet como

$$z_{l+1} = z_l + \mathcal{F}_l(z_l) \tag{3.2}$$

Sendo d a dimensão da entrada, então $z_l \in \mathbb{R}^d$ é a entrada da camada l e $\mathcal{F}_l : \mathbb{R}^d \to \mathbb{R}^d$ é uma função diferenciável que preserva a dimensão de z_l. A diferença $z_{l+1} - z_l$ pode ser interpretada como a discretização da derivada *z'(l)* com um passo $\Delta l = 1$. Considerando $\Delta l \to 0$, pode então ser determinado

$$\lim_{\Delta l \to 0} \frac{Zl + \Delta l - Zl}{\Delta l} = \frac{dz(l)}{dl} = \mathcal{F}(z(l), l) \tag{3.3}$$

de forma que a dinâmica dentre as camadas da ResNet pode ser parametrizada como uma ODE. Com essa formulação (Equação 3.3), um vetor de dados x de entrada pode ser mapeado para suas características a serem aprendidas pela rede neuronal como resultado (a saída $\phi(x)$ da ResNet) com a solução do seguinte PVI para uma camada T (profundidade máxima). Ou seja, o valor $z(T)$ corresponde à saída aprendida pelo modelo.

$$\frac{dz(l)}{dl} = \mathcal{F}(z(l), l), \;\; z(0) = x \tag{3.4}$$

Além disso, é possível definir também esse modelo NODE para problemas de regressão e classificação. Até aqui, as ODEs foram usadas para mapear os dados de entrada $x \in \mathbb{R}^d$ a um conjunto de características ou uma representação $\phi(x) \in \mathbb{R}^d$ por meio da rede neuronal. No entanto, para problemas de classificação e regressão, é interessante definir um modelo $\mathbb{R}^d \to \mathbb{R}$. Isso foi apresentado por Lin e Jegelka (2018) para ResNets. Sendo a NODE uma função $\mathcal{G} : \mathbb{R}^d \to \mathbb{R}$ com $\mathcal{G}(x) = \mathcal{L}(\phi(x))$, onde $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ é uma função linear e $\phi : \mathbb{R}^d \to \mathbb{R}^d$ é um mapeamento dos dados de entrada para um conjunto de características ou representação. Isso pode ser visualizado no modelo simplificado da Figura 3.1 mostrando uma camada ODE seguida por uma camada linear.

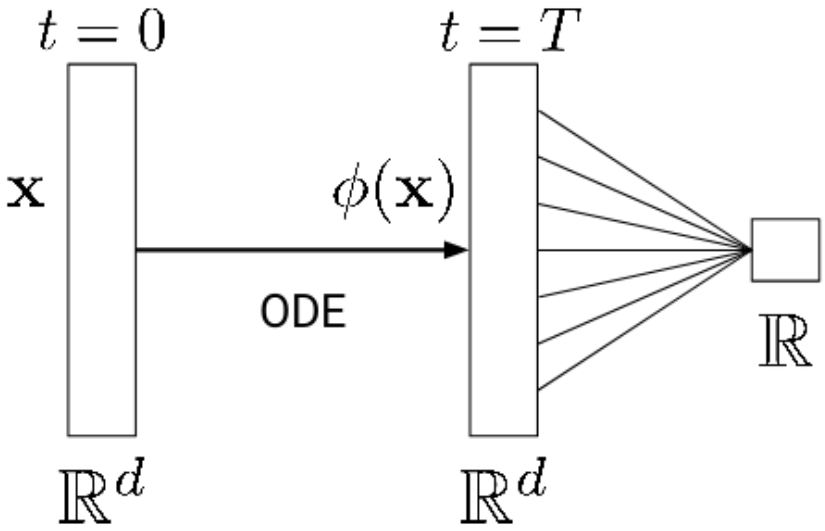

Figura 3.1: Diagrama simplificado apresentando a arquitetura de uma NODE. Imagem de Dupont et al. (2019).

Ou seja, a entrada $z(T)$ da camada $l = T$ é a solução da ODE para um PVI começando na camada de entrada $z(0)$, que equivale aos dados de entrada x. Como apresentado em Chen et al. (2018), isso é computado numericamente por um resolvedor de ODEs que retorna, com a tolerância a erro dada, o resultado para a camada determinada (o valor l fornecido).

Definindo o modelo NODE nesse formato, o resolvedor de ODEs é considerado uma caixa preta e representa o maior custo computacional da rede neuronal por ser usado tanto no cálculo da saída quanto para o treinamento da NODE.

3.2 TREINAMENTO

A maior dificuldade técnica para o treinamento de redes neuronais de profundidade contínua é calcular a diferenciação reversa necessária para executar a retropropagação, como foi apresentado em Chen et al. (2018). Com a utilização do resolvedor de ODEs, é trivial aplicar os resultados intermediários calculados durante as transformações da entrada ao avançar na NODE para fazer a diferenciação. No entanto, essa prática implica em um uso enorme de memória e introduz erro numérico adicional.

Utilizando um resolvedor de ODEs como caixa preta, é proposto em Chen et al. (2018) a computação dos gradientes utilizando o *adjoint sensitivity method* apresentado em Pontryagin (1987). Essa abordagem computa os gradientes resolvendo uma segunda ODE aumentada de forma reversa, que pode ser utilizada para todos os resolvedores de ODEs.

Para treinamento da NODE, é definida uma função escalar de perda $\mathcal{P}$ a ser otimizada, que calcula o erro da camada de saída $l = T$ e recebe como entrada o resultado de um resolvedor de ODEs:

$$\mathcal{P}(z(T)) = \mathcal{P}(z(l_0) + \int_{l_0}^{T} \mathcal{F}(z(l), l, W)dl) = \mathcal{P}(ODE\_Solver(z(l_0), \mathcal{F}, l_0, T, W)) \tag{3.5}$$

Para otimizar $\mathcal{P}$, é necessário obter o gradiente em relação aos parâmetros W. O primeiro passo é determinar como o gradiente da função de perda depende de *z(l)* em cada etapa. Esse valor, denominado *adjoint a* é definido como:

$$a(l) = \frac{\partial \mathcal{P}}{\partial z(l)} \tag{3.6}$$

Como afirmado em Chen et al. (2018) e demonstrado detalhadamente no Apêndice B do artigo citado, a dinâmica do valor *adjoint* é dada por outra ODE, que pode ser interpretada como um análogo imediato da regra da cadeia:

$$\frac{da(l)}{dl} = -a(l)^{\mathsf{T}} \frac{\partial \mathcal{F}(z(l), l, W)}{\partial z} \tag{3.7}$$

Dessa forma, $a(l_0)$ pode ser computado com uma nova chamada ao resolvedor de ODEs. O resolvedor deve ser executado de forma reversa, solucionando um PVI em que *a(T)* é o valor inicial que corresponde ao gradiente já calculado de $\mathcal{P}$ em relação a *z(T)*. Resolver essa ODE tem como obstáculo o conhecimento do valor de *z(l)* para cada l durante toda a trajetória. No entanto, *z(l)* também pode ser calculado de forma reversa juntamente com o *adjoint*, resolvendo o PVI a partir do valor final *z(T)* já obtido.

Para calcular o gradiente de $\mathcal{P}$ em relação aos parâmetros W em l_0 é necessário determinar como o gradiente da função de perda depende de W durante toda a trajetória. Sendo a_W o *adjoint* relacionado aos parâmetros e denotado por

$$a_W(l) = \frac{\partial \mathcal{P}}{\partial W} \tag{3.8}$$

Pode ser determinada mais uma ODE análoga ao que foi apresentado na Equação 3.7 descrita na forma

$$\frac{da_W(l)}{dl} = -a(l)^\intercal \frac{\partial \mathcal{F}(z(l), l, W)}{\partial W} \tag{3.9}$$

que permite determinar $a_W(l)$ para a camada inicial l_0 como a solução do PVI resolvido a partir do valor já conhecido $a_W(T)$.

Vale ressaltar que os vetores-produto Jacobianos $a(l)^\intercal \frac{\partial \mathcal{F}}{\partial z}$ e $a(l)^\intercal \frac{\partial \mathcal{F}}{\partial W}$, cujos valores são necessários para a solução, podem ser calculados eficientemente por diferenciação automática com custo computacional semelhante ao demandado pelo cálculo de $\mathcal{F}$. Além disso, as integrais que são necessárias para resolver z, a a_W podem ser computadas em uma única execução do resolvedor de ODEs, com a concatenação desses valores em um só vetor. A aplicação modelada dessa forma tem escala linear em relação ao tamanho do problema, baixo uso de memória e controle explícito do erro numérico. No entanto, muitos resolvedores de ODEs fornecem a opção de determinar o valor *z(l)* em múltiplas etapas. Quando $\mathcal{P}$ depende desses valores intermediários, a resolução reversa com o *adjoint sensitivity method* deve ser quebrada em uma sequência de cálculos separados. Esse processo é ilustrado na Figura 3.2.

O algoritmo apresentado em Chen et al. (2018) exemplifica como a computação em uma única chamada do resolvedor de ODEs pode ser feita. Em essência, o algoritmo engloba em um vetor todos os valores a serem calculados simultaneamente. Para melhor notação, é definida uma função aumentada f_{aug}, que reúne as três ODEs a serem resolvidas, descrita na forma

$$f_{aug}(z(l), a(l), l, W) = \left[\mathcal{F}(z(l), l, W), -a(l)^\intercal \frac{\partial \mathcal{F}}{\partial z}, -a(l)^\intercal \frac{\partial \mathcal{F}}{\partial W} \right] \tag{3.10}$$

Então os valores $[z(l_0), a(l_0), a_W(l_0)]$ da retropropagação são concatenados em um único vetor para poderem ser computados simultaneamente em uma só chamada do resolvedor de ODEs, que irá retornar $\left[z(l_0), \frac{\partial \mathcal{P}}{\partial z(l_0)}, \frac{\partial \mathcal{P}}{\partial W} \right]$ para a camada inicial l_0. A notação final determinando a chamada feita ao resolvedor de ODEs, utilizando $z(T)$, $a(T)$ e $a_W(T)$ como valores iniciais, é descrita como

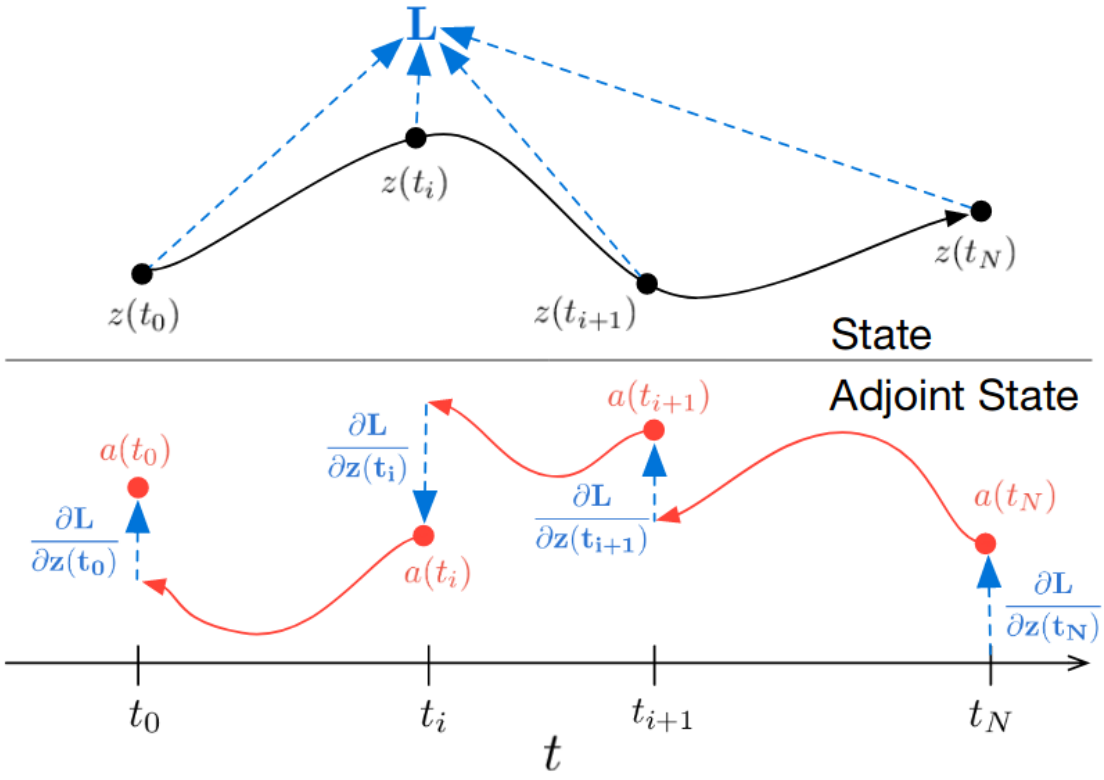

Figura 3.2: Resolução reversa de uma ODE. O *adjoint sensitivity method* soluciona uma ODE aumentada na trajetória reversa das camadas da rede. A cada etapa, o sistema aumentado contém o valor z da camada e a perda $\mathcal{P}$ relativa a camada. No diagrama, é utilizado t como valor da camada atual, sendo t_0 a camada inicial. Imagem de Chen et al. (2018).

$$ODE\_Solver([z(T), a(T), a_W(T)], f_{aug}(z(l), a(l), l, W), T, l_0, W) \tag{3.11}$$

de forma que as integrais que são necessárias para obter z, a e a_W já são computadas durante a execução do resolvedor de ODEs. Para controle explícito do erro numérico, basta acrescentar o erro máximo como um parâmetro do resolvedor de ODEs tanto no cálculo de *z(T)* quanto na computação da retropropagação.

3.3 RESULTADOS

Nesta seção, serão apresentados os resultados obtidos em Chen et al. (2018) através de experimentos investigando o treinamento de NODEs para aprendizado supervisionado. Como o desempenho depende diretamente do resolvedor de ODEs, foi utilizado o método de Adams implícito implementado em LSODE, como descrito em Radhakrishnan e Hindmarsh (1993), e em VODE, como apresentado em Brown et al. (1989), a partir do pacote **scipy.integrate**. A implementação do *adjoint sensitivity method* foi feita no *framework* de Python **autograd** apresentado em Maclaurin et al. (2015). A dinâmica das camadas escondidas e suas respectivas

derivadas foram calculadas em uma GPU utilizando o **Tensorflow** a partir das chamadas feitas pelos resolvedores de ODEs mencionados, implementados em Fortran, que foram executados por meio de código feito em **autograd**.

Como critério de comparação, foram feitos experimentos em uma ResNet com 6 blocos residuais, como mostrado em He et al. (2016), fazendo a substituição dos blocos ResNet do modelo por uma variante NODE. Para a verificação do impacto da utilização do resolvedor de ODEs, o modelo NODE também foi testado executando a retropropagação dos gradientes pelo método de Runge-Kutta. O testes foram feitos na base MNIST apresentada em LeCun et al. (1998). A Tabela 3.1 apresenta o percentual de erro, o número de parâmetros e os custos na computação dos testes.

	Test Error	# Params	Memory	Time
1-Layer MLP†	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(\tilde{L})$

Tabela 3.1: Desempenho obtido pela execução dos modelos na base MNIST. A tabela apresenta o modelo 1-Layer MLP original de LeCun et al. (1998), a ResNet de 6 blocos residuais, a NODE RK-Net utilizando retropropagação com Runge-Kutta e a NODE ODE-Net especificada em Chen et al. (2018). O valor L denota o número de camadas na ResNet e $\tilde{L}$ representa o número de avaliações executadas pelo resolvedor de ODEs durante uma passagem pelo modelo, que pode ser interpretado como um número implícito de camadas (uma aproximação da profundidade da NODE). Tabela de Chen et al. (2018).

Os experimentos mostram que ambas as NODEs apresentam aproximadamente o mesmo desempenho das ResNets, mas utilizando menos parâmetros. Além disso, a ODE-Net apresenta uso de memória de apenas $O(1)$, valor inferior ao encontrado nos outros modelos.

Outro fator interessante é a possibilidade de controle de erro na utilização das NODEs. Isso ocorre porque os resolvedores de ODEs permitem, de forma aproximada, delimitar a tolerância de erro do resultado encontrado em relação à solução real. Modificar essa tolerância altera diretamente o comportamento do modelo. Na Figura 3.3 são apresentados: (a) o controle de erro; (b) a relação entre o tempo gasto na propagação e o número de avaliações efetuadas pelo resolvedor de ODEs; (c) a relação da quantidade de avaliações entre a propagação e a retropropagação na rede; (d) a quantidade de avaliações durante a propagação em relação ao treinamento.

Os resultados de (b) mostram que que o tempo gasto pela propagação é proporcional ao número de avaliações feitas pelo resolvedor de ODEs, permitindo conciliar a acurácia e o custo computacional a partir da configuração da tolerância a erro. Além disso, é possível treinar o modelo com baixa tolerância a erro e alto custo computacional, mas testar com baixa acurácia gastando pouco tempo. A relação encontrada em (c) sugere que o *adjoint sensitivity method* é mais eficiente tanto em uso de memória quanto em custo computacional, já que a retropropagação

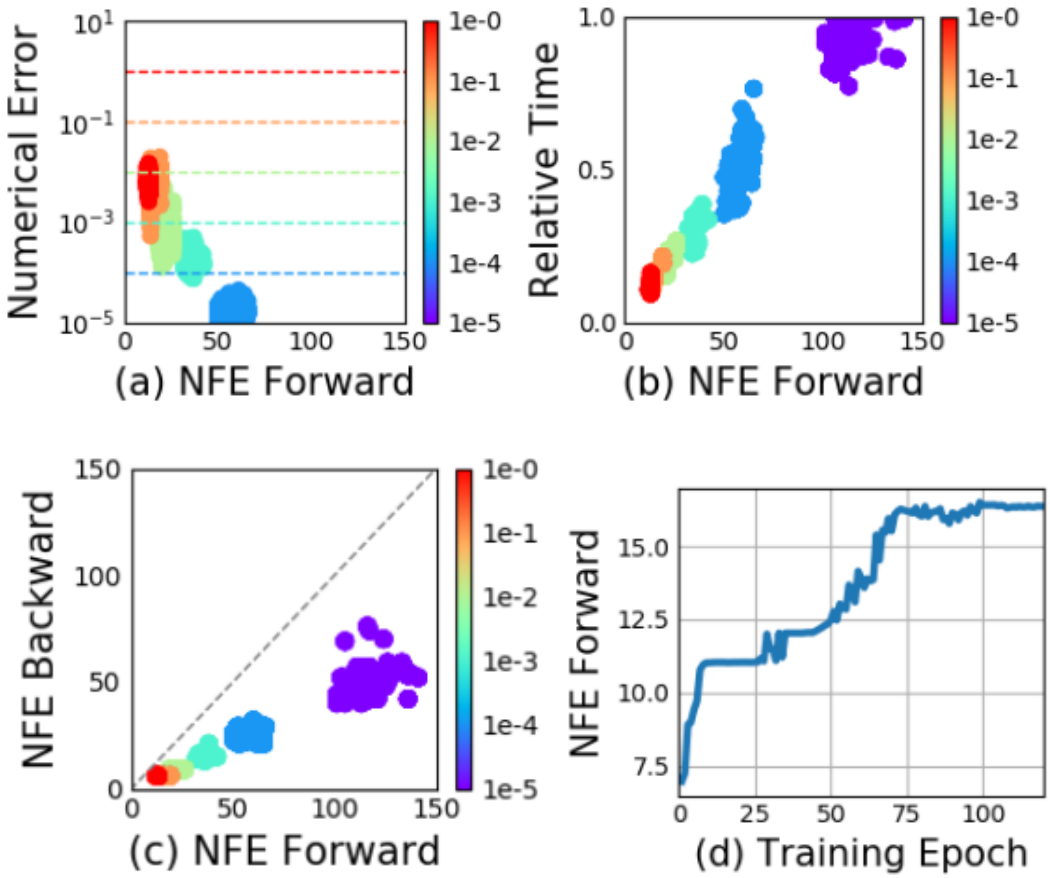

Figura 3.3: Gráficos apresentando as estatísticas relacionadas ao treinamento da NODE. Em (a) é apresentada a relação entre a tolerância definida para o erro no resolvedor de ODEs (eixo vertical) e o erro numérico encontrado (cor correspondente). Já em (b) é constatada a relação entre o tempo gasto pela propagação e o número de avaliações feitas pelo resolvedor de ODEs. Pode ser percebido em (c) que o número de avaliações do resolvedor de ODEs na retropropagação é aproximadamente metade do que é feito na propagação. Finalmente, em (d) é mostrado o número de avaliações feitas pelo resolvedor de ODEs em relação ao progresso do treinamento da NODE. Imagem de Chen et al. (2018).

convencional implica em calcular os gradientes para todas as avaliações feitas pelo resolvedor de ODEs durante a propagação. O gráfico (d) permite uma análise relacionada ao conceito de profundidade no modelo NODE. Como não é claro como definir quantas camadas uma NODE contém, devido à natureza da estrutura do modelo, um valor relacionado que pode ser utilizado é o número de avaliações feitas pelo resolvedor de ODEs. É constatado que esse número aumenta durante o treinamento, sugerindo uma adaptação da rede com a complexidade do modelo (a ocorrência de uma aproximação crescente entre o modelo e a função a ser aprendida).

3.4 ESCOPO E LIMITAÇÕES

Além das vantagens das NODEs apresentadas em relação a ResNets e outros modelos convencionais, esse modelo também mostra grande potencial em problemas específicos, como o caso de *Continuous Normalizing Flows* ou o aprendizado, interpolação e extrapolação de séries temporais apresentados por Chen et al. (2018). No entanto, foi demonstrado por Dupont et al. (2019) que esse modelo aprende preservando a topologia do espaço das entradas utilizadas no

treinamento, podendo aprender apenas conjuntos de características que sejam homeomorfos à entrada. Dois espaços topológicos são ditos homeomorfos caso exista uma função entre ambos que seja contínua, invertível e que sua inversa seja contínua. Essa propriedade e suas características são apresentados de forma clara em Zeeman (1966) e em Liu (2012), possibilitando perceber que essa limitação também proíbe qualquer transformação que faça algum corte ou buraco na superfície. Isso implica que existem classes de funções que NODEs não podem aprender adequadamente. Enquanto essas limitações, obstáculos computacionais resultantes e demonstrações pertinentes são extensivamente detalhados em Dupont et al. (2019), nessa seção será analisado um simples exemplo, de apenas 1 dimensão, que ilustra o problema encontrado nas NODEs.

Seja $g_{1d} : \mathbb{R} \to \mathbb{R}$ uma função de forma que $g_{1d}(-1) = 1$ e $g_{1d}(1) = -1$. As trajetórias ao mapear -1 para 1 e 1 para -1 obrigatoriamente deverão se cruzar. No entanto, trajetórias de ODEs não podem se interceptar, de forma que uma ODE aprendida na NODE não pode representar $g_{1d}(x)$. Essa intuição é a base da prova detalhada em Dupont et al. (2019) e é uma observação que serve como núcleo para muitos exemplos que demonstram as limitações das NODEs. A dinâmica da função exemplo $g_{1d}(x)$ é apresentada na Figura 3.4 mais adiante. Os gráficos demonstram o comportamento esperado no treinamento das NODEs e verificado pelos experimentos feitos tanto para uma função identidade quanto para a função exemplo $g_{1d}(x)$.

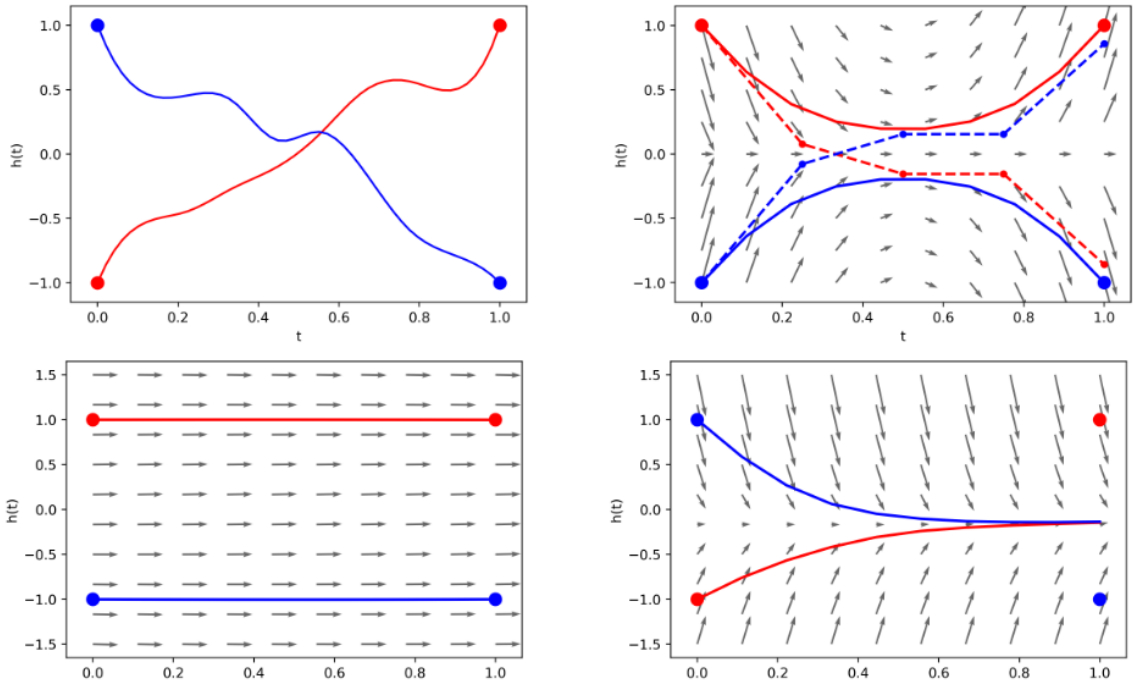

Figura 3.4: Gráficos apresentando os resultados dos experimentos feitos investigando as limitações das NODEs. O valor $h(t)$ corresponde à saída da camada t, permitindo a visualização das transformações realizadas pelas camadas escondidas na entrada. No topo à esquerda, são apresentadas as trajetórias contínuas ao mapear -1 para 1 (vermelho) e 1 para -1 (azul). Já no topo à direita, as soluções computadas para as ODEs estão em linhas cheias, enquanto as soluções computadas pelo método de Euler (correspondente às ResNets) são mostradas em linhas tracejadas. Como mostrado, a discretização permite que as trajetórias possam se interceptar. Logo abaixo, são apresentados os campos vetoriais resultantes do treinamento do modelo NODE para uma função identidade (à esquerda) e para a função $g_{1d}(x)$ exemplo (à direita). Imagem de Dupont et al. (2019).

Como pode ser percebido, o modelo NODE pode aprender a função identidade facilmente, mas não é capaz de representar $g_{1d}(x)$ de forma adequada. Inclusive, devido à impossibilidade de interceptação das trajetórias, a NODE devolve zero para todas as entradas como forma de minimizar o erro.

Já que as NODEs podem ser interpretadas como um equivalente contínuo das ResNets, é interessante considerar porque as ResNets podem representar $g_{1d}(x)$ e as NODEs não. O motivo dessa discrepância é justamente porque o funcionamento das ResNets é uma discretização de ODEs, permitindo assim que as trajetórias façam saltos discretos entre si. Dessa forma, o erro relacionado ao uso de passos discretos nas ResNets permite que as trajetórias possam se interceptar, possibilitando a representação de mais funções.

Um último exemplo pode ser apresentado brevemente para ilustrar como as limitações das NODEs não se restringem a casos de interceptação de trajetórias ou apenas para 1 dimensão. Dada uma classe de funções, onde $0 < r_1 < r_2 < r_3$ e seja $g_d : \mathbb{R}^d \to \mathbb{R}$ para qualquer dimensão d, de forma que

$$\begin{cases} g_d(x) = -1 \text{ se } \|x\| \leqslant r_1 \\ \\ g_d(x) = 1 \text{ se } r_2 \leqslant \|x\| \leqslant r_3 \end{cases} \tag{3.12}$$

sendo $\|x\|$ a norma euclidiana de x. Uma ilustração dessa função para o caso $d = 2$ é apresentada na Figura 3.5 (a) em que a função mapeia todos os pontos dentro do círculo azul para -1 e todos os pontos do ânulo vermelho para 1, apresentando de forma clara o espaço dos dados de entrada.

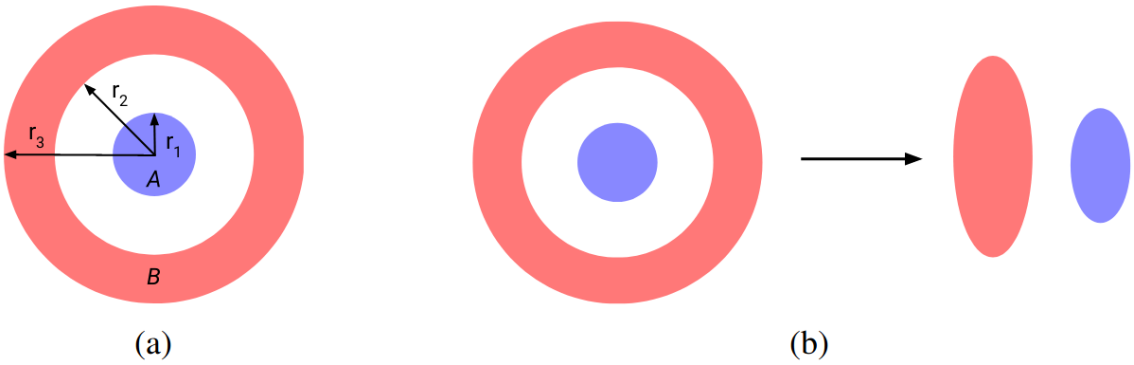

Figura 3.5: (a) Diagrama representando a função exemplo $g_d(x)$ para $d = 2$. b) Um exemplo de mapeamento $\phi(x)$ a partir dos dados de entrada para características necessárias para a representação de $g_d(x)$. As NODEs não são capazes de aprender esse mapeamento $\phi(x)$. Imagem de Dupont et al. (2019).

Conforme demonstrado em Dupont et al. (2019), a intuição para constatar que NODEs não são capazes de representar $g_d(x)$ é bastante simples. A camada é capaz de mapear a entrada para -1 e 1 adequadamente quando o modelo a ser aprendido $\phi(x)$ para os pontos azuis e vermelhos for linearmente separável. Como a região azul está cercada pela região vermelha, quaisquer trajetórias a serem aprendidas a partir dos dados, que não são linearmente separáveis,

vai exigir que as trajetórias tenham que se interceptar e isso não é possível para as NODEs. O mapeamento $\phi(x)$ das NODEs preserva a topologia do espaço da entrada, com um comportamento de homeomorfismo, conforme mostrado em Younes (2010). Isso implica que a NODE pode apenas fazer uma deformação contínua no espaço da entrada, não sendo capaz de romper uma região conectada. Essa limitação não permite a representação de $g_d(x)$, conforme o resultado mostrado na Figura 3.5 (b), já que a execução de algum corte ou buraco que é necessária para fazer a separação das duas regiões não é permitida a $\phi(x)$. É apresentado em Dupont et al. (2019) que é possível aprender a representação de $g_d(x)$ de forma adequada executando um aumento nas dimensões no espaço da entrada, permitindo assim a preservação da topologia do espaço de entrada a partir de um mapeamento que não apresente cruzamento de trajetórias.

4 EXPERIMENTOS

Para uma melhor compreensão e a avaliação mais rigorosa do modelo NODE, foram feitos diversos experimentos utilizando como base a implementação disponibilizada em Chen et al. (2018) para o problema de aprendizado da identificação de dígitos. Para isso, foi utilizada a base MNIST inicialmente para a comparação entre o desempenho obtido pelo modelo NODE nos experimentos feitos e os resultados relatados em Chen et al. (2018). Além disso, foram feitos experimentos com a base UFPR-AMR, utilizada em Laroca et al. (2019), e com a base UFPR-ALPR, apresentada em Laroca et al. (2018). Para essas duas bases, foram utilizados os modelos NODE e ResNet para a comparação dos resultados, a partir da utilização de várias camadas Convolucionais no intuito de aprender uma representação de características a partir das imagens de entrada. Todos os experimentos foram executados em um computador com CPU Intel i7-6700 3.40GHz e 16 GB de RAM.

A arquitetura do modelo NODE é detalhada na Tabela 4.1, utilizando um bloco NODE para aprender a representação a ser utilizada para a classificação logística na camada Linear. Esse bloco NODE, detalhado na Tabela 4.2, consiste basicamente de operações de convolução e é utilizado pelo resolvedor de ODEs para executar quantas computações forem necessárias para atingir a tolerância definida.

#	Camada	Kernel/Stride	Input	Output
0	Convolucional	3x3/1	28x28x3	26x26x64
1	Group Normalization		26x26x64	26x26x64
2	ReLU		26x26x64	26x26x64
3	Convolucional	4x4/2	26x26x64	13x13x64
4	Group Normalization		13x13x64	13x13x64
5	ReLU		13x13x64	13x13x64
6	Convolucional	4x4/2	13x13x64	6x6x64
7	Bloco NODE		6x6x64	6x6x64
8	Group Normalization		6x6x64	6x6x64
9	ReLU		6x6x64	6x6x64
10	Adaptive Avg Pool	Global	6x6x64	1x1x64
11	Flatten		1x1x64	64x1x1
12	Linear		64x1x1	{10,36}x1x1

Tabela 4.1: Tabela detalhando as camadas utilizadas no modelo NODE. A entrada da primeira camada Convolucional recebe os três canais das imagens coloridas. A saída da camada Linear é de 10 classes para a base UFPR-AMR (dígitos numéricos) e de 36 para a UFPR-ALPR (dígitos numéricos e letras).

O modelo ResNet tem a sua arquitetura detalhada na Tabela 4.3 e utiliza seis blocos ResNet para aprender a representação das características. Novamente, essa representação é então

#	Camada	Kernel/Stride	Input	Output
0	Group Normalization		6x6x64	6x6x64
1	ReLU		6x6x64	6x6x64
2	Convolucional	3x3/1	6x6x64	6x6x64
3	Group Normalization		6x6x64	6x6x64
4	Convolucional	3x3/1	6x6x64	6x6x64
5	Group Normalization		6x6x64	6x6x64

Tabela 4.2: Tabela detalhando as camadas do bloco NODE utilizado dentro do modelo.

utilizada para a classificação logística na camada Linear. O bloco ResNet também consiste basicamente de operações de convolução e é descrito na Tabela 4.4.

#	Camada	Kernel/Stride	Input	Output
0	Convolucional	3x3/1	28x28x3	26x26x64
1	Group Normalization		26x26x64	26x26x64
2	ReLU		26x26x64	26x26x64
3	Convolucional	1x1/2	26x26x64	13x13x64
4	Convolucional	3x3/2	13x13x64	7x7x64
5	Group Normalization		7x7x64	7x7x64
6	Convolucional	3x3/1	7x7x64	7x7x64
7	Group Normalization		7x7x64	7x7x64
8	ReLU		7x7x64	7x7x64
9	Convolucional	1x1/2	7x7x64	4x4x64
10	Convolucional	3x3/2	4x4x64	2x2x64
11	Group Normalization		2x2x64	2x2x64
12	Convolucional	3x3/1	2x2x64	2x2x64
[13..18]	Blocos ResNet		2x2x64	2x2x64
19	Group Normalization		2x2x64	2x2x64
20	ReLU		2x2x64	2x2x64
21	Adaptive Avg Pool	Global	2x2x64	1x1x64
22	Flatten		1x1x64	64x1x1
23	Linear		64x1x1	{10,36}x1x1

Tabela 4.3: Tabela detalhando as camadas utilizadas no modelo ResNet. As camadas [13..18] consistem em seis blocos ResNet. A entrada da primeira camada Convolucional recebe os três canais das imagens coloridas. A saída da camada Linear é de 10 classes para a base UFPR-AMR (dígitos numéricos) e de 36 para a UFPR-ALPR (dígitos numéricos e letras).

#	Camada	Kernel/Stride	Input	Output
0	Group Normalization		2x2x64	2x2x64
1	ReLU		2x2x64	2x2x64
2	Convolucional	3x3/1	2x2x64	2x2x64
3	Group Normalization		2x2x64	2x2x64
4	Convolucional	3x3/1	2x2x64	2x2x64

Tabela 4.4: Tabela detalhando as camadas do bloco ResNet utilizado dentro do modelo.

Para o treinamento, a função de perda utilizada é a *CrossEntropyLoss*, que é bastante útil para casos de classificação com muitas classes. Essa função executa uma operação de *SoftMax* no vetor de escores resultante da camada Linear, comparando então a distância entre ele e o vetor *one-hot* da classe correta.

4.1 EXPERIMENTO COM A BASE MNIST

O repositório indicado em Chen et al. (2018) foi instalado e então foi utilizada a base dados MNIST, disponibilizada pelo pacote **torchvision**, para a avaliação dos resultados do modelo. O número de parâmetros utilizado no experimento foi 208266, sendo consistente com o valor de 0.22 M que foi relatado no artigo. Além disso, o menor erro encontrado de 0.30% é ainda menor que o erro de 0.42% apresentado no artigo. Outro resultado verificado é a rápida convergência durante o treinamento. O gráficos apresentados na Figura 4.1 mostram o comportamento do erro em relação às épocas de treinamento:

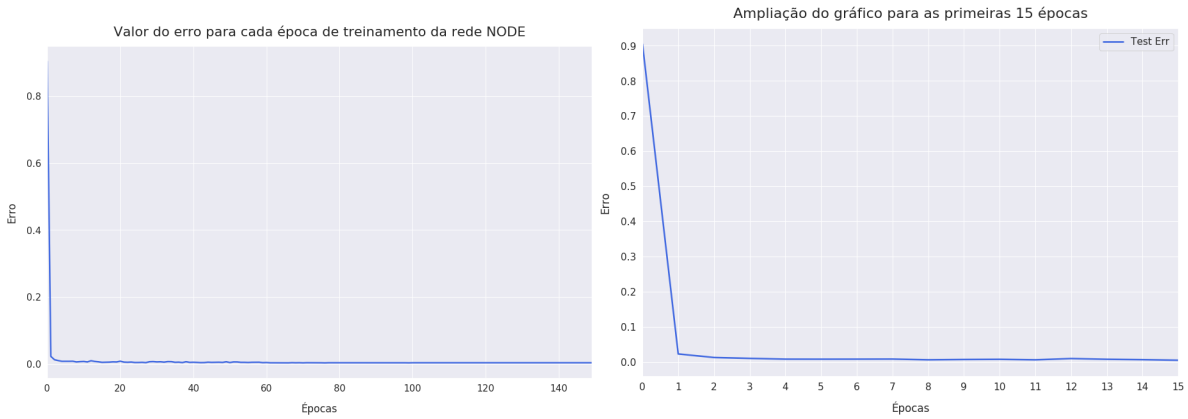

Figura 4.1: Erro calculado durante as épocas de treinamento do modelo. No gráfico da esquerda, são apresentadas todas as épocas. Já no gráfico da direita, é feita uma ampliação para melhor visualização da rápida convergência já nas primeiras épocas do treinamento.

4.2 EXPERIMENTO COM A BASE UFPR-AMR

Conforme apresentado em Laroca et al. (2019), *Automatic Meter Reading* (AMR) se refere ao registro automático do consumo de energia elétrica, gás e água, feito tanto para o monitoramento quanto para a execução da cobrança desses serviços. A base da dados UFPR-AMR contém 2000 imagens de medidores, como é exemplificado na Figura 4.2:

Utilizando os rótulos fornecidos, os cinco dígitos do contador de energia elétrica de cada medidor foram recortados com *padding* de 4 pixels. Dessa forma, foram obtidos 4000 dígitos para treinamento, 4000 para validação e 2000 para teste. Além disso, as imagens

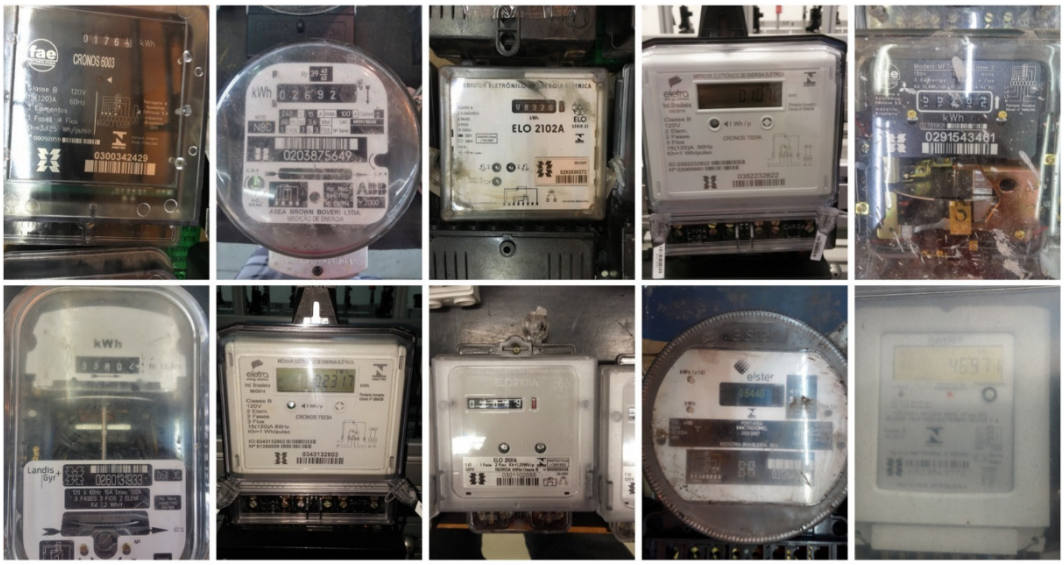

Figura 4.2: Alguns exemplos dos diversos medidores contidos na base de dados utilizada. A UFPR-AMR contém tanto contadores analógicos quanto digitais, além de apresentar fotografias com reflexos sobre determinados locais do medidor, incluindo os contadores. Imagem de Laroca et al. (2019)

foram redimensionadas para um formato 28x28, mesmas dimensões da MNIST, para agilizar o processamento.

Foram feitos experimentos sobre a base resultante desse pré-processamento com os modelos NODE e ResNet para fins de comparação, com alguns dos resultados apresentados na Tabela 4.5. O modelo NODE atingiu uma acurácia maior do que a alcançada pela ResNet, mas apresenta um tempo de processamento muito superior. Devido ao uso do *adjoint method*, o custo computacional pela utilização do resolvedor de ODEs foi ainda mais alto.

	NODE	ResNet
Parâmetros	209418	602634
Erro	2.90%	3.85%
Batch Time Avg	21.96s	4.89s

Tabela 4.5: Tabela apresentando o resultado dos experimentos com os modelos NODE e ResNet

Outro aspecto notável é a diferença entre os modelos do comportamento do erro em relação às épocas de treinamento. Embora a NODE tenha obtido um erro menor ao fim do treinamento, é possível perceber no gráfico apresentado na Figura 4.3 uma instabilidade maior na convergência do modelo durante as 30 primeiras épocas.

Finalmente, é importante avaliar o número de computações executadas pelo resolvedor de ODEs durante o treinamento. Conforme apresentado em Chen et al. (2018), esse número durante a propagação à frente pode ser interpretado como a profundidade das camadas ODEs da rede neuronal. O número de computações executadas na propagação à frente é considerado NFE-F e as executadas na retropropagação é NFE-B. Como é observado na Figura 4.4, o NFE-F

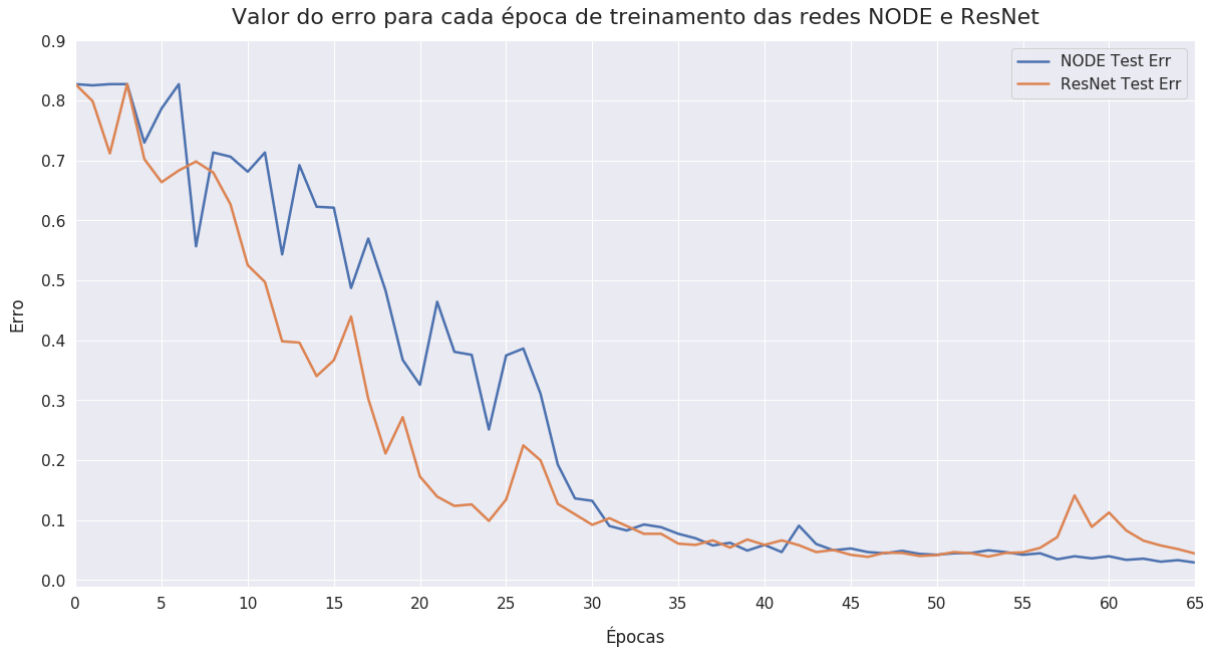

Figura 4.3: Erro calculado durante as épocas de treinamento dos modelos NODE e ResNet. Embora apresente maior instabilidade na convergência inicialmente, o modelo NODE obteve o menor erro.

aumenta durante as épocas de treinamento, indicando um aprofundamento do bloco ODE da rede neuronal, que pode ser associado à uma adaptação que aumenta a complexidade do modelo. Além disso, é visível que o NFE-B se mantém em aproximadamente metade do NFE-F. Isso indica que o *adjoint sensitivity method* não é só mais eficiente em relação à memória, mas também apresenta melhor custo computacional, já quer não é necessário executar a retropropagação executando os cálculos para cada computação feita pelo resolvedor de ODEs durante a propagação à frente.

Como é utilizado o NFE-F como uma referência para a profundidade do bloco ODE e o número de parâmetros do modelo NODE é aproximadamente um terço daquele utilizado na ResNet, é visível novamente a eficiência da NODE em relação ao uso de parâmetros para o treinamento. Mesmo com uma base mais complexa que a MNIST como a UFPR-AMR, contendo mais ruídos e certo desbalanceamento, o modelo NODE mostrou manter o mesmo comportamento e características descritos em Chen et al. (2018).

4.3 EXPERIMENTO COM A BASE UFPR-ALPR

A identificação de dígitos também faz parte da tarefa de ALPR (*Automatic License Plate Recognition*), que consiste no reconhecimento automático de placas de veículos. Essa aplicação prática é muito pertinente para cobrança automática de pedágios, manutenção das leis de trânsito, controle de acesso a espaços privados e monitoramento do tráfego nas estradas. A tarefa de ALPR é descrita detalhadamente e a base de dados UFPR-ALPR é apresentada em Laroca et al. (2018). Essa base contém 30 imagens de 150 veículos, sendo as fotografias feitas de dentro de um

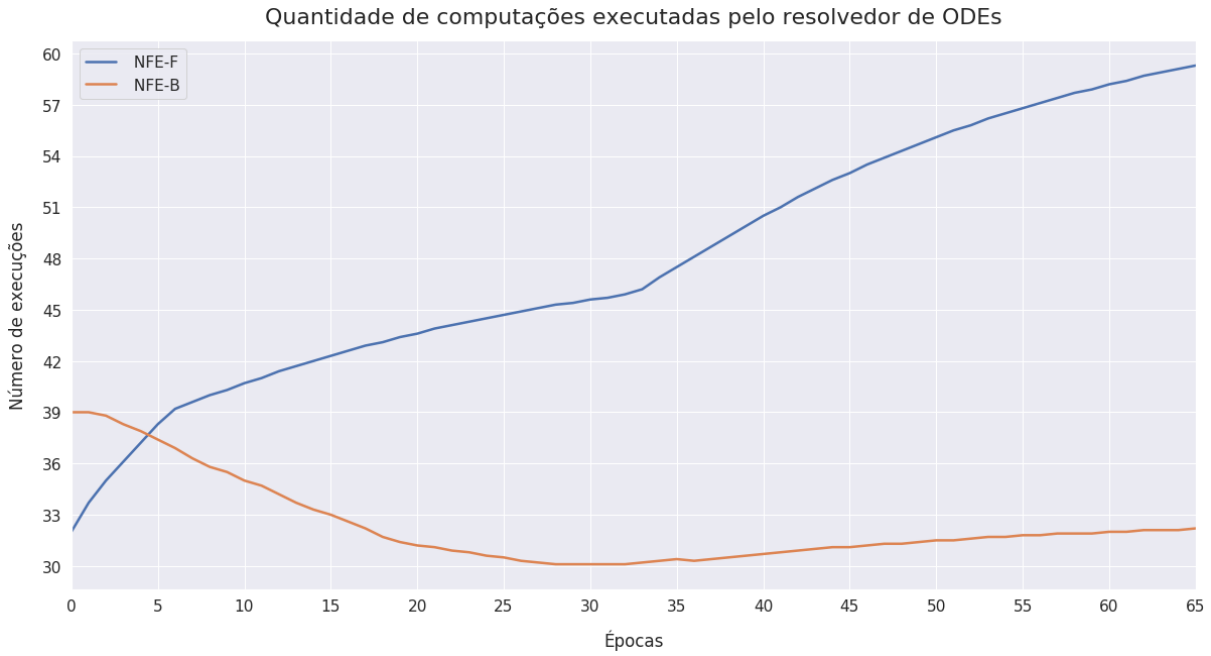

Figura 4.4: Gráfico apresentando o número de computações executadas pelo resolvedor de ODEs na propagação à frente (NFE-F) e as executadas na retropropagação (NFE-B). Já que não é claro como definir a profundidade do bloco ODE, é utilizado o NFE-F como referência para isso.

veículo transitando em um ambiente urbano. As fotografias são resultado de um vídeo gravado por 1 segundo de cada veículo em uma taxa de 30 frames por segundo (FPS). Dessa forma, foram obtidas fotografias contendo placas dos mais diversos veículos em variadas condições de luz, como é exemplificado na Figura 4.5. A partir dos rótulos fornecidos na base, os sete dígitos das placas foram recortados com *padding* de 1 pixel. Como as fotografias foram feitas com considerável distância dos veículos, vale ressaltar que as imagens de dígitos resultantes desse recorte apresentam uma resolução bem inferior ao resultado observado na base UFPR-AMR. Finalmente, foram obtidos 12600 dígitos para treinamento, 12600 para validação e 6300 para teste. Novamente, as imagens foram redimensionadas para 28x28 com o objetivo de acelerar o processamento.

Ao analisar a base de imagens de dígitos resultante, foi constatado que o número de casos para cada letra era bem inferior ao observado para os dígitos numéricos. Além disso, o número de imagens para casos de certas classes de letras se mostrou uma quantidade pequena demais para o treinamento adequado de uma rede neuronal. Isso se deve à diferença de número de classes entre esses dois tipos de dígitos, já que são 26 de letras e apenas 10 de dígitos numéricos. Após efetuar um experimento com o modelo NODE sobre a base nessa condição, foi efetuado um aumento nos dados nas bases de treinamento e validação (*Data Augmentation*), triplicando a quantidade de imagens de letras. Isso foi feito com a aplicação de um filtro de nitidez e com a transformação das imagens originais em escala de cinza, como é apresentado na Figura 4.6:

Figura 4.5: Exemplos das imagens de diversos veículos contidas na base de dados UFPR-ALPR. É possível perceber as variações de cores, tamanho das placas e das condições de iluminação. As placas foram borradas por questões de privacidade. Imagem de Laroca et al. (2018)

Figura 4.6: Um exemplo demonstrando os efeitos utilizados após o recorte para aumentar a quantidade de dados de letras nas bases de treinamento e validação. O restante da placa foi borrado por questões de privacidade.

A partir disso, foram feitos experimentos utilizando as bases resultantes com os modelos NODE e o ResNet. Dessa forma, foi possível comparar o desempenho do modelo NODE em relação à técnica de aumento de dados além da comparação dos resultados em uma mesma base contra o modelo ResNet. Os resultados desses experimentos são apresentados na Tabela 4.6 para cada um dos modelos.

	NODE	NODE Data Aug	ResNet Data Aug
Parâmetros	211108	211108	604324
Erro	16.08%	14.25%	16.97%
Batch Time Avg	27.59s	27.17s	5.74s

Tabela 4.6: Tabela apresentando o resultado dos experimentos com os modelos NODE e ResNet

Da mesma forma que foi feito na base UFPR-AMR, a diferença entre o comportamento do erro em relação às épocas de treinamento foi avaliada para os três experimentos, como é apresentando na Figura 4.7. O modelo NODE se beneficiou bastante do aumento dos dados de letras nas bases de treinamento e validação, apresentando convergência maior durante o treinamento quando comparada com o mesmo modelo aplicado sobre a base original. Além disso, a convergência foi mais rápida que a ResNet e o menor erro obtido foi menor, mesmo utilizando os dois modelos sobre a mesma base de dados aumentada.

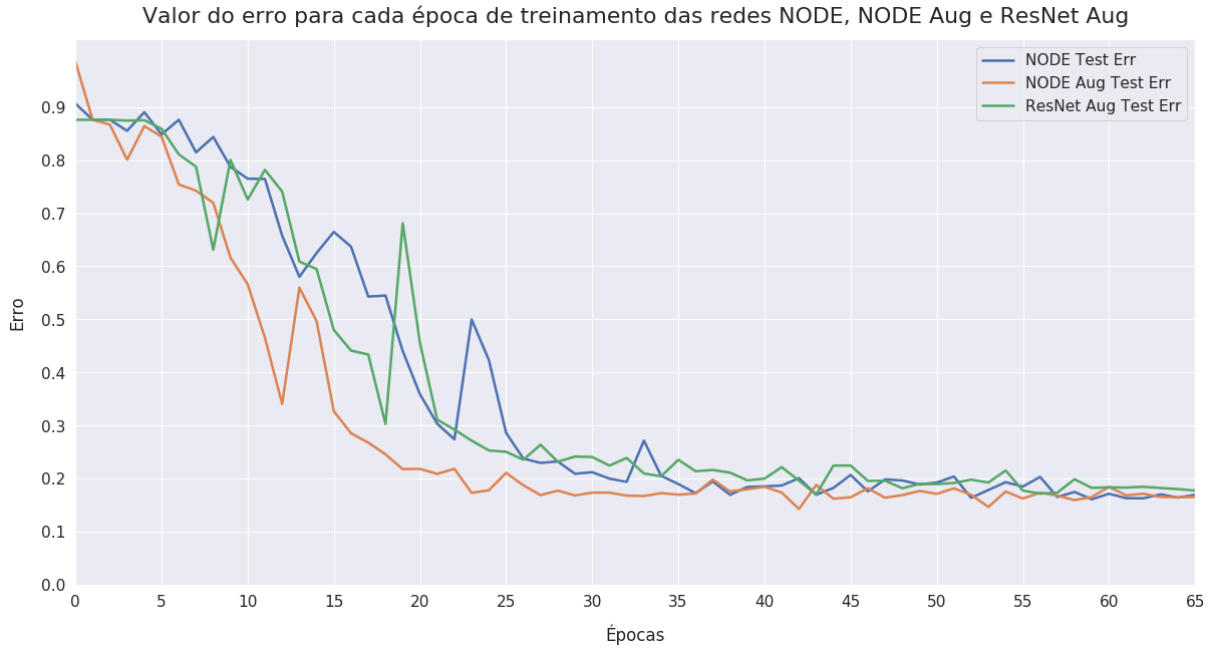

Figura 4.7: Erro calculado durante as épocas de treinamento do modelo NODE sobre a base de dados original, do modelo NODE sobre a base de dados aumentada e da ResNet sobre a base de dados aumentada. O modelo NODE teve desempenho melhor quando utilizado na base de dados aumentada e atingiu o menor erro.

Finalmente, é avaliado novamente os NFE-F e NFE-B durante as épocas de treinamento do modelo NODE, utilizando para isso o experimento feito sobre a base de dados aumentada. Outra vez, o NFE-F cresce durante as épocas de treinamento, sugerindo um aumento na

profundidade do bloco ODE da rede neuronal e na complexidade do modelo. Isso pode ser observado na Figura 4.8. Da mesma forma que ocorreu na base UFPR-AMR, o NFE-B é muito inferior ao NFE-F, confirmando a eficiência do *adjoint sensitivity method* em relação à memória e ao custo computacional afirmada em Chen et al. (2018).

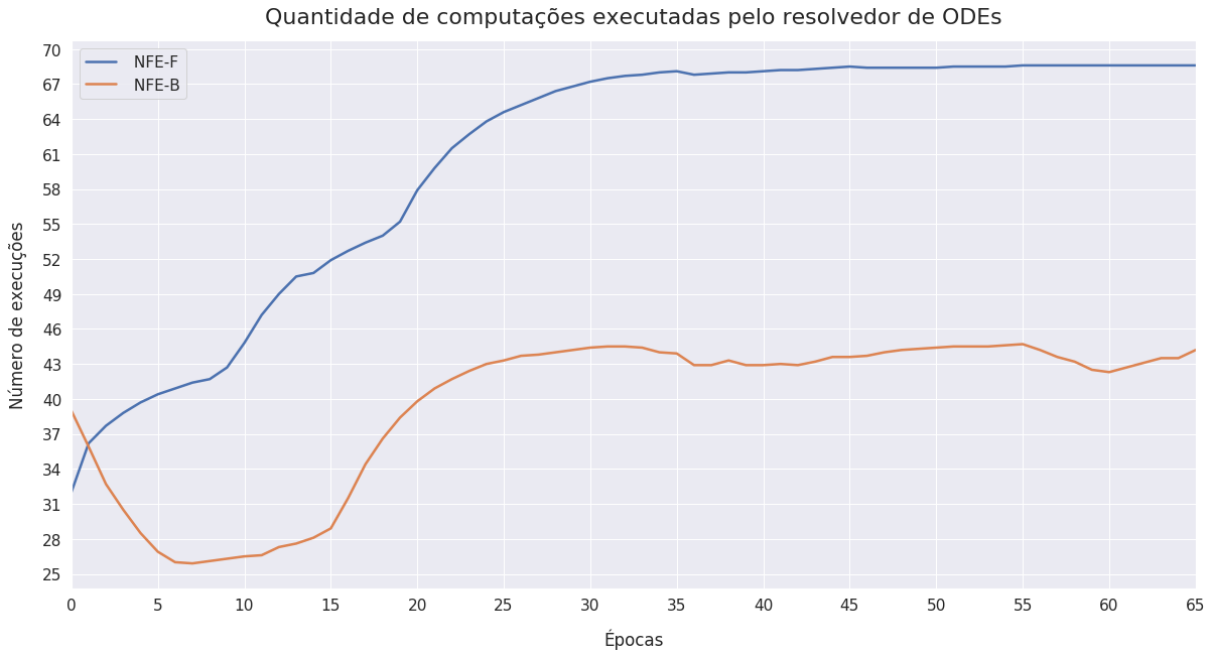

Figura 4.8: Erro calculado durante as épocas de treinamento do modelo NODE sobre a base de dados original, do modelo NODE sobre a base de dados aumentada e da ResNet sobre a base de dados aumentada. O modelo NODE teve desempenho melhor quando utilizado na base de dados aumentada e atingiu o menor erro.

Ao interpretar o NFE-F como referência para a profundidade do bloco ODE, pode ser concluído que o modelo ODE treinado com a base UFPR-ALPR resultou em uma rede neuronal com mais camadas e mais complexa que a rede resultante do treinamento na base UFPR-AMR. Isso é coerente com o fato das placas de veículos apresentarem 36 classes de dígitos contra apenas 10 classes de números dos medidores de energia. Além disso, as fotografias dos medidores de energia resultaram em dígitos recortados com melhor resolução e contraste.

5 CONSIDERAÇÕES FINAIS

Ao discorrer sobre o cenário, os avanços e o comportamento das ResNets, fica clara a importância do desenvolvimento desse modelo para o avanço das redes neuronais profundas, eliminando o problema da degradação em redes com muitas camadas e viabilizando o ganho teórico de acurácia atribuído ao aumento da profundidade. Como afirmado em Szegedy et al. (2016), a introdução de conexões residuais em conjunção com uma arquitetura convencional produziu desempenho equivalente ao estado da arte no ILSVRC 2015, além de acelerar o treinamento quando aliada a outros modelos de redes neuronais.

Aproveitando os avanços das ResNets, as NODEs são um modelo recente que apresenta enorme potencial para resolver problemas complexos melhor ou com desempenho equivalente a outras técnicas semelhantes. A sua estrutura é baseada em uma observação matemática natural em relação ao modelo de aprendizado residual profundo e utiliza soluções sofisticadas em relação às adaptações necessárias para um treinamento eficiente. Isso fica claro ao usar como exemplo a solução proporcionada pelo *adjoint sensitivity method* para a execução da retropropagação no treinamento da NODE, um método descrito por Pontryagin já em 1962 e que descarta a necessidade de armazenar as computações executadas pelo resolvedor de ODEs durante a propagação à frente. Como foi dito pela equipe do artigo na NeurIPS 2018, em uma entrevista para a Synced, a matemática é para sempre. O modelo NODE é mais um caso dentro da área de *Deep Learning* onde descobertas matemáticas antigas são fundamentais para os avanços no estado da arte, mostrando a enorme relevância da matemática independentemente do tempo.

5.1 TRABALHOS FUTUROS

Após o estudo aprofundado de toda a teoria embasando o modelo NODE e a análise dos resultados obtidos pela comunidade científica, além dos experimentos feitos com a classificação de dígitos, ainda restam várias aplicações e limitações do modelo NODE que foram apresentadas em Chen et al. (2018) a serem exploradas. Comparar o desempenho nos experimentos em outras bases com um modelo aumentado, como proposto em Dupont et al. (2019), também pode trazer melhor entendimento sobre as limitações das *Neural Ordinary Differential Equations*. Felizmente, ambos artigos fornecem os códigos-fonte utilizados para os experimentos, incluindo a documentação e orientações para a instalação dos requisitos necessários para a execução, nos repositórios abaixo:

- `github.com/rtqichen/torchdiffeq`

- `github.com/EmilienDupont/augmented-neural-odes`

É importante ressaltar que o modelo NODE é bastante flexível, permitindo a utilização das mais diversas configurações de camadas dentro do bloco NODE. Isso propicia enorme potencial para pesquisar os resultados de diferentes blocos NODE para variados problemas e bases de dados, além da possível associação com outras arquiteturas de redes neuronais.

5.2 CONCLUSÃO

O modelo de *Neural Ordinary Differential Equations* é uma adaptação adequada das ResNets, obtendo bom proveito a partir das semelhanças dentre a dinâmica das camadas das ResNets e o processo iterativo dos métodos numéricos dos resolvedores de ODEs. O obstáculo da execução da retropropagação é ultrapassado de forma sofisticada e eficiente pelo uso do *adjoint sensitivity method* dentro da função aumentada contida no resolvedor de ODEs. A arquitetura NODE atinge resultados equivalentes às ResNets em relação a acurácia, além de apresentar potencial de convergência mais rápida, logo de início, durante o treinamento. Com o uso da rede NODE, é possível ajustar o custo computacional a partir da tolerância de erro dada ao resolvedor de ODEs, obter custo constante em relação ao uso da memória e diminuir consideravelmente a quantidade de parâmetros necessários para o treinamento do modelo. No entanto, as NODEs só são capazes de aprender funções que resultam em saídas que são homeomorfas ao espaço dos valores de entrada. Essa limitação não ocorre nas ResNets devido à dinâmica discreta das suas camadas. Dessa forma, as NODEs apenas são viáveis dentro de um escopo determinado, onde o seu uso proporciona várias vantagens e possibilita a aproximação de sistemas contínuos de forma inédita nos outros modelos de redes neuronais. Os experimentos feitos nas bases UFPR-AMR e UFPR-ALPR reforçam as afirmações feitas sobre o modelo NODE em Chen et al. (2018), demonstrando resultados compatíveis ao que foi relatado no artigo com a utilização do modelo na MNIST. Além disso, foi possível constatar com clareza a enorme diferença de tempo de processamento entre o modelo NODE e as ResNets, implicando que o uso das *Neural Ordinary Differential Equations* só é viável em casos de estruturas com grande poder computacional e em aplicações que não necessitem de modelos rápidos, onde o tempo de pŕocessamento não seja crítico. Finalmente, o modelo NODE mostra potencial para aprender representações de alta complexidade, por possuir a possibilidade de utilizar quantas camadas forem necessárias durante o treinamento.

REFERÊNCIAS

Bengio, Y., Simard, P. e Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Brown, P. N., Byrne, G. D. e Hindmarsh, A. C. (1989). Vode: A variable-coefficient ode solver. *SIAM J. Sci. Stat. Comput.*, 10(5):1038–1051.

Canziani, A., Paszke, A. e Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678.

Chen, T. Q., Rubanova, Y., Bettencourt, J. e Duvenaud, D. (2018). Neural ordinary differential equations. *CoRR*, abs/1806.07366.

Dupont, E., Doucet, A. e Teh, Y. W. (2019). Augmented neural odes. *ArXiv*, abs/1904.01681.

Glorot, X. e Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Em Teh, Y. W. e Titterington, M., editores, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 de *Proceedings of Machine Learning Research*, páginas 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Haber, E. e Ruthotto, L. (2017). Stable architectures for deep neural networks. *CoRR*, abs/1705.03341.

Hairer, E., Norsett, S. P. e Wanner, G. (1993). *Solving Ordinary Differential Equations I. Nonstiff Problems*. Springer, Berlin, 2nd rev. ed. 1993. corr. 3rd printing edition. ID: unige:12346.

He, K., Zhang, X., Ren, S. e Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

He, K., Zhang, X., Ren, S. e Sun, J. (2016). Identity mappings in deep residual networks. *CoRR*, abs/1603.05027.

Hochuli, A. (2019). *Abordagens livres de segmentação para reconhecimento automático de cadeias numéricas manuscritas utilizando aprendizado profundo*. Tese de doutorado, UFPR - Universidade Federal do Paraná, Curitiba - Brasil. 90 pgs.

Krizhevsky, A., Sutskever, I. e Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90.

Kutta, W. (1901). Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeit. Math. Phys.*, 46:435–53.

Lambert, J. D. (1991). *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*. John Wiley & Sons, Inc., New York, NY, USA.

Laroca, R., Barroso, V., Diniz, M. A., Gonçalves, G. R., Schwartz, W. R. e Menotti, D. (2019). Convolutional neural networks for automatic meter reading. *Journal of Electronic Imaging*, 28:1–14.

Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S., Gonçalves, G. R., Schwartz, W. R. e Menotti, D. (2018). A robust real-time automatic license plate recognition based on the YOLO detector. Em *International Joint Conference on Neural Networks (IJCNN)*, páginas 1–10.

LeCun, Y., Bottou, L., Bengio, Y. e Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323.

Lin, H. e Jegelka, S. (2018). Resnet with one-neuron hidden layers is a universal approximator. *CoRR*, abs/1806.10909.

Liu, J. (2012). An extention of zeeman's an introduction to topology. Bachelor's thesis, University of California, Santa Barbara - Estados Unidos. 22 pgs.

Lu, Y., Zhong, A., Li, Q. e Dong, B. (2017). Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *CoRR*, abs/1710.10121.

Maclaurin, D., Duvenaud, D. e Adams, R. P. (2015). Autograd: Effortless gradients in numpy. Em *ICML 2015 AutoML Workshop*.

Nair, V. e Hinton, G. (2010). Rectified linear units improve restricted boltzmann machines. volume 27, páginas 807–814.

Pontryagin, L. (1987). *Mathematical Theory of Optimal Processes*. Classics of Soviet Mathematics. Taylor & Francis.

Radhakrishnan, K. e Hindmarsh, A. C. (1993). Description and use of lsode, the livemore solver for ordinary differential equations.

Runge, C. (1895). Ueber die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178.

Simonyan, K. e Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints*, página arXiv:1409.1556.

Szegedy, C., Ioffe, S. e Vanhoucke, V. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261.

Weinan, E. (2017). A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5:1–11.

Younes, L. (2010). *Shapes and Diffeomorphisms*. Applied Mathematical Sciences. Springer Berlin Heidelberg.

Zeeman, E. C. (1966). *An introduction to topology; the classification theorem for surfaces*. Mathematics Institute, University of Warwick, Coventry. 51 pgs.