Wilson Guasti Junior

Soluções de equações diferenciais via redes neurais artificiais

Vitória, ES

2021

Wilson Guasti Junior

Soluções de equações diferenciais via redes neurais artificiais

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Isaac Pinheiro dos Santos

Vitória, ES

2021

Wilson Guasti Junior
Soluções de equações diferenciais via redes neurais artificiais/ Wilson Guasti Junior. – Vitória, ES, 2021-
104 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Isaac Pinheiro dos Santos

Dissertação de Mestrado – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Programa de Pós-Graduação em Informática, 2021.

1. Palavra-chave1. 2. Palavra-chave2. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. Soluções de equações diferenciais via redes neurais artificiais

CDU 02:141:005.7

Wilson Guasti Junior

Soluções de equações diferenciais via redes neurais artificiais

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Trabalho aprovado. Vitória, ES, 26 de fevereiro de 2021:

Prof. Dr. Isaac Pinheiro dos Santos
PPGI/UFES
Orientador

Profa. Dra. Lucia Catabriga
PPGI/UFES
Examinador

Prof. Dr. Leonardo Muniz de Lima
IFES
Examinador Externo

Prof. Dr. Cassio Machiaveli Oishi
FCT/UNESP
Examinador Externo

Vitória, ES
2021

Dedico a Deus, minha esposa Mônica e a minha filha Maria Clara, pelo apoio, força, amparo e compreensão.

Agradecimentos

Agradeço primeiramente a Deus, sem o qual nada é possível.

À minha esposa Mônica e minha filha Maria Clara por todo o apoio, ajuda e compreensão, pois nunca desistiram de mim, foram minha grande motivação e sempre acreditaram no meu potencial.

Ao meu orientador Isaac Pinheiro do Santos por sua enorme ajuda, paciência, companheirismo e amizade, o estudo nunca foi tão agradável.

Aos meus pais Wilson e Olga pelo apoio incondicional, também para minha avó Lídia, cujo amor sempre será lembrado.

Aos meus sogros, Antônio e Marlene, por me apoiarem nesta jornada.

Também a todos os meus familiares, cuja amizade sempre será lembrada.

Ao Professor Orivaldo de Lira Tavares por ter acreditado em mim.

À UFES pelo ensino gratuito de qualidade, ao PPGI, sem os quais essa dissertação dificilmente poderia ter sido realizada.

"A vantagem de ter péssima memória é divertir-se
muitas vezes com as mesmas coisas
boas como se fosse a primeira vez.
(Friedrich Nietzsche)

Resumo

O estudo de equações diferenciais exerce um importante papel em vários campos da ciência e tecnologia, através da modelagem de problemas do mundo real. Como grande parte dos modelos matemáticos descritos por equações diferenciais (ordinárias e parciais) não possui solução analítica, método numéricos, como diferenças finitas e elementos finitos, são amplamente utilizados para resolvê-lo. Recentemente, muitos estudos têm sido dedicados na aplicação de redes neurais artificiais profundas, conhecidas como *deep learning*, no processo de solução de equações diferenciais, com resultados promissores.

O objetivo deste trabalho é explorar o uso de modelos de aprendizagem profunda (*deep learning*) baseados em redes neurais artificiais *feedforward* na solução de equações diferenciais ordinárias e parciais. A rede neural foi implementada usando a linguagem Python, com a biblioteca Tensorflow. Aplicamos essa metodologia na solução de dois problemas de valor inicial, no problema de Poisson (bidimensional), em dois problemas transientes (equações do calor e da onda) e em um problema singularmente pertubado unidimensional (equação de convecção-difusão) para avaliar a qualidade das soluções obtidas. Algumas comparações com os métodos numéricos clássicos, Euler, Runge-Kutta, diferenças finitas e elementos finitos, são apresentadas.

Palavras-chaves: Equações Diferenciais. Redes Neurais Artificiais. Tensorflow. Aprendizagem Profunda.

Abstract

The study of differential equations plays an important role in several fields of science and technology, through the modeling of real-world problems. As most of the mathematical models described by differential equations (ordinary and partial) do not have an analytical solution, numerical methods, such as finite differences and finite elements, are widely used to solve it. Recently, many studies have been dedicated to the application of deep artificial neural networks, known as deep learning, in the solution of differential equations, with promising results.

The aim of this work is to explore the use of artificial neural networks feedforward in the solution of ordinary and partial differential equations. The neural network was implemented using the Python language, with the Tensorflow library. We applied this methodology in the solution of two initial value problems, in the Poisson problem (two-dimensional), in two unsteady problems (heat and wave equations) and in a singularly perturbed one-dimensional problem (convection-diffusion equation) to evaluate the quality of the solutions obtained . Some comparisons with classical numerical methods, such as Euler, Runge-Kutta, finite differences and finite elements, are presented.

Keywords: Differential Equations. Artificial Neural Networks. Tensorflow. Deep Learning.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

BFGS	Broyden-Fletcher-Goldfarb-Shanno
DA	Diferenciação Automática
EDO	Equação Diferencial Ordinária
MDF	Método de Diferenças Finitas
MEF	Método de Elementos Finitos
PMC	Perceptron Multicamadas
RNA	Redes Neurais Artificiais
RPROP	Resilient Propagation

método de diferenças finitas (MDF) ou o método de elementos finitos (MEF)

Sumário

1 Introdução

Equações diferenciais desempenham um importante papel em vários campos da ciência e tecnologia, modelando problemas do mundo real de grande interesse na sociedade. De fato, uma variedade de problemas presentes na engenharia, física, economia, biologia, epidemiologia, etc. pode ser descrito através de equações diferenciais, ordinárias ou parciais (EDOs/EDPs). Para citar alguns exemplos, que fogem do senso comum, destacamos a detecção de falsificações de artes, o diagnóstico de diabetes e outras doenças, o aumento da porcentagem de tubarões presentes no mar Mediterrâneo durante a primeira guerra mundial e a disseminação de epidemias, como o Covid-19 (BRAUN, 1975; QUARTERONI, 2009; WANG et al., 2020; GRAVE et al., 2021).

Em geral, soluções exatas (analíticas) de modelos matemáticos descritos por equações diferenciais não são fáceis de serem obtidas. Neste caso, devemos recorrer aos métodos numéricos que fornecem soluções aproximadas para esses problemas. Existem vários tipos de métodos numéricos para resolver EDOs e EDPs, tais como, Runge-Kutta, diferenças finitas, elementos finitos, volumes finitos, etc. (SMITH, 1985; ROOS; STYNES; TOBISKA, 1996; QUARTERONI, 2009). Esses métodos são baseados na discretização do domínio de definição do problema em um número finito de pontos ou elementos, de forma que a solução aproximada é obtida nesses pontos. Soluções aproximadas em pontos fora dos pontos discretizados são obtidas via um mecanismo de intepolação. Uma desvantagem dos métodos numéricos, como os métodos de diferenças finitas e elementos finitos (clássico), é a necessidade de usar malhas bem refinadas para obtenção de melhores soluções. O custo computacional dessa estratégia pode crescer muito, inviabilizando o seu uso em problemas com dimensão elevada. Isso está associado ao fenômeno conhecido como maldição da dimensionalide (*curse of dimensionality*), segundo o qual a complexidade computacional de um determinado problema cresce exponencialmente como uma função da dimensão desse problema (BELLMAN, 1957; E; HAN; JENTZEN, 2020; LEAKE; MORTARI, 2020). Segundo Leake e Mortari (2020), o número de elementos no método de elementos finitos cresce exponencialmente com o número de dimensões, de forma que esse método é eficiente para problemas definidos em domínios com dimensão pequena.

As redes neurais artificiais (RNAs) são modelos de aprendizagem de máquinas, que funcionam como os neurônios do cérebro humano. RNAs têm recebido muita atenção nos últimos anos, devido ao seu sucesso em inúmeras aplicações importantes, como reconhecimento de padrões, correlações em conjuntos de dados, etc. Matematicamente, uma RNA pode ser definida como um grafo direcionado, onde os vértices representam os neurônios e as arestas

representam as conexões entre os neurônios. Existem muitas variantes de redes neurais que se diferem em relação a sua arquitetura (como os neurônios são conectados). Um dos modelos mais simples RNAs é a rede *feedforward*, também conhecida como rede *perceptron* multicamadas (PMC). Os termos *redes neurais profundas* (*deep neural network*) e *aprendizagem profunda* (*deep learning*) referem ao uso de redes neurais com muitas camadas ocultas, ou escondidas, em problemas de aprendizagem de máquinas (*machine learning*). Uma das vantagens em adicionar camadas ocultas em uma rede neural é a redução exponencial do custo computacional em algumas aplicações e o decréscimo exponencial da quantidade de dados de treinamento necessários no processo de aprendizagem da rede (BENGIO; SIMARD; FRASCONI, 1994).

Um dos trabalhos precussores na utilização de redes neurais artificiais na solução de equações diferenciais foi apresentado em (LAGARIS; LIKAS; FOTIADIS, 1998). Neste trabalho, os autores utilizaram uma rede neural *feedforward* para resolver EDOs e EDPs em domínios retangulares. O método consiste em minimizar uma função custo, descrita pelo resíduo da equação nos pontos de amostragem, e dada em função dos parâmetros (pesos) da rede. O treinamento da rede consistiu na minimização da função custo, em função dos pesos, utilizando o método Quase-Newton, Broyden-Fletcher-Goldfarb-Shanno (BFGS). A estratégia de resolver equações diferenciais via redes neurais já tinha sido utilizada nos trabalhos descritos em (LEE; KAND, 1990; MEADE; FERNANDEZ, 1994).

A solução aproximada de equações diferenciais via RNA tem várias vantagens, que dependem do modelo de RNA utilizado. Em comparação com os métodos numéricos convencionais, destacamos as seguintes propriedades das soluções obtidas via RNA (LAGARIS; LIKAS; FOTIADIS, 1998; PARISI; MARIANI; LABORDE, 2003), que motivam o seu uso:

a) a solução é contínua e diferenciável no domínio de definição do problema;

b) a solução é obtida no domínio todo, eliminando a necessidade do uso de interpolação numérica;

c) a complexidade computacional não aumenta consideravelmente com o aumento no número de pontos de discretização[1] e com o número de dimensões envolvidas no problema;

d) o método pode ser aplicado na solução de EDOs ou EDPs lineares, bem como não lineares e transientes;

e) a propagação de erros de arredondamento, comum nos métodos numéricos clássicos, não afeta a solução obtida via redes neurais.

[1] No contexto de redes neurais, chamamos de pontos de colocação ou de amostragem.

Uma outra motivação importante em utilizar redes neurais para resolver equações diferenciais é sua propriedade de aproximar funções contínuas. O *teorema da aproximação universal* (CYBENKO, 1989; HORNIK; STINCHCOMBE; WHITE, 1989) afirma que RNAs *feedforward* podem aproximar qualquer função contínua em um conjunto compacto[2], com uma precisão arbitrária. Segundo Hornik, Stinchcombe e White (1989), as redes neurais multicamadas *feedforward* são aproximadores universais, então, pelo menos em teoria, essas redes podem fornecer uma solução para qualquer equação diferencial, com uma margem de erro arbitrariamente pequena. Desde então, um número grande de trabalhos que exploram o uso de redes neurais na solução de problemas de valores de contorno e iniciais tem sido publicado na literatura. A seguir descrevemos alguns deles.

Lagaris, Likas e Papageorgiou (2000) apresentaram um método para resolver problemas de valor de contorno em domínio com fronteiras irregulares. Aarts e Veer (2001) apresentaram um método baseado em RNA para resolver problemas de valor de contorno e inicial, envolvendo EDPs. O método proposto leva em conta o fato de que as redes *feedforward* com entrada múltipla, uma única camada oculta e uma única saída (linear) sem limiares são capazes de aproximar funções arbitrárias e suas derivadas. Informações sobre a EDP e suas condições inicial e de contorno são incorporadas na estrutura do modelo. Jianyu et al. (2003) apresentaram um método para resolver EDPs elípticas, onde as funções de ativação dos neurônios da camada oculta são funções de bases radiais (*radial basis functions* - RBF), cujos parâmetros são ajustados através do método da Descida Máxima. Segundo os autores, a estratégia de treinamento do método economiza tempo computacional e espaço de memória. Shirvany, Hayati e Moradian (2009) apresentaram um método basedo em RNA *perceptron* multicamadas e funções de bases radiais para resolver EDOs e EDPS. Os autores aplicaram o método na solução da equação não linear de Schrodinger. Além disso, o método apresentou bons resultados na solução de problemas com solução exata, mostrando rápida convergência e pouco uso de memória quando comparado com os métodos de Runge-Kutta e elementos finitos. Beidokhti e Malek (2009) propuseram um método híbrido basedo em RNA, técnicas de otimização e métodos de colocação para determinar uma solução aproximada, em uma forma analítica e fechada, para um sistema geral de equações diferenciais parciais dependentes do tempo, com condições inicial e de contorno arbitrárias. Outros métodos baseados em redes neurais são descritos em (PARISI; LABORDE, 2001; PARISI; MARIANI; LABORDE, 2003; MCFALL; MAHAN, 2009; BAYAMANI; EFFATI; KERAYECHIAN, 2011; KUMAR; YADAV, 2011; RUDD, 2013).

Nos últimos quatro anos várias metodologias baseadas em redes neurais profundas (*deep learning*) foram apresentadas na literatura para resolver problemas de valores de contorno

[2] Em dimensão finita, um conjunto compacto é um conjunto fechado e limitado.

e inicial (CHAUDHARI et al., 2017; RAISSI; KARNIADAKIS, 2018; AGGARWAL, 2018; BECK et al., 2018; DOCKHORN, 2019; GUO et al., 2020; SAMANIEGO et al., 2020). Redes neurais profundas tem se mostrado eficiente na solução de problemas envolvendo grande quantidade de dados e informações, inclusive apresentando soluções para o problema da maldição da dimensionalidade (*curse of dimensionality*), no contexto de EDPs, como abordado em (HAN; JENTZEN; WEINAN, 2017; JENTZEN; KRUSE; NGUYEN, 2020). Em geral, os problemas modelados no contexto da mecânica do contínuo podem depender de até quatro variáves independentes (três espaciais e uma temporal), mas existem problemas que podem ser descritos em termos de centenas de variáveis independentes, como àqueles que aparecem na área de finanças, envolvendo as equações de Black-Sholes REFERENCIA. Em particular, Sirignano e Spiliopoulos (2018) apresentam um método baseado em redes neurais profundas, livre de malha, chamado *Deep Galerkin Method* (DGM), para resolver EDPs definidas em domínios de alta ordem. Nesse trabalho, em cada iteração do processo de aprendizagem, um conjunto diferente de pontos do domínio (e da fronteira) é utilizado. O algoritmo desenvolvido foi capaz de resolver problemas com até 200 dimensões. Como sabemos, as técnicas numéricas clássicas, como métodos de diferenças finitas e elementos finitos, não são efetivas na solução de problemas que dependem de muitas variáveis independentes, devido ao grande número de pontos nodais nas malhas que discretizam o domínio.

O uso de redes neurais profundas para resolver equações diferenciais possui vantagens e desvantagens, em relação aos métodos numéricos clássicos, como descrito em (DOCKHORN, 2019). Neste trabalho, o autor se baseou na solução do problema de Poisson e nas equações de Navier-Stokes estacionária, usando uma rede *feedforward*, o método BFGS para otimizar a função custo, funções de ativação sigmoidal em combinação com processo de inicialização de Xavier, para . Como vantagens, ele destaca (i) pequenas RNAs são capazes de aproximar bem a solução de equações diferenciais, (ii) facilidade de implementação do método, usando o algoritmo *backpropagation*, (iii) extensão para domínios de dimensões maiores; os pontos de treinamento podem ser escolhidos aleatoriamente, (iv) método livre de espaços de funções; para resolver as equações de Navier-Stokes usando o método de elementos finitos, devemos escolher espaços de funções que satisfazem a condição *inf-sup* (HUGHES, 1987), (v) incorporação de informações de sensores (ruídos) no modelo, adicionando um termo na função custo. Como desvantagens, ele destaca (i) *convergência*: devido à questões do processo de otimização, não foi possível mostrar empiricamente o decaimento do erro, com uma certa taxa de convergência, com o aumento do tamanho da rede neural[3] (ii) *tempo de execução*: o tempo de execução utilizando o método de otimização BFGS foi consideravelmente maior do que àquele obtido com os métodos

[3] Taxas de convergência teórica para redes neurais na solução de equações diferenciais é um problema em aberto.

numéricos clássicos, (iii) *escalabilidade*: dificuldade em prever o comportamento do método para problemas mais complicados, por exemplo, equações de Navier-Stokes para resolver escoamentos tridimensionais turbulentos, (iv) aleatoriedade dos dados iniciais: diferentes inicializações aleatórias conduzem a diferentes resultados, enquanto que, em geral, os métodos numéricos clássicos são determinísticos. O autor sugere que mais investigação é necessária para um melhor entendimento desse comportamento do método.

O sucesso das redes neurais profundas (*deep learning*) em importantes tarefas de aprendizagem pode estar relacionado ao desenvolvimento de ferramentas computacionais eficientes, como as bibliotecas Tensorflow (ABADI et al., 2016), PyTorch (PASZKE et al., 2019) outras, que fornecem mecanismos para computação de alto desempenho e diferenciação automática. O Tensorflow, desenvolvido em 2015 pelo Google, é um framework com código aberto compatível com Python e amplamente utilizado para o desenvolvimento de redes neurais profundas. Essa biblioteca foi idealizada para ser flexível, eficiente, extensível e portátil. Muito embora o TensorFlow tenha amplo suporte para aplicações em aprendizagem de máquina, também é igualmente adequado para realizar cálculos matemáticos complexos (ABRAHAMS et al., 2016). Existem outras opções utilizadas em aprendizagem profunda, tais como: Theano[4](TEAM et al., 2016), Keras[5](CHOLLET et al., 2015) e Torch[6](COLLOBERT; BENGIO; MARITHOZ, 2002). No TensorFlow, as derivadas são calculadas utilizando algoritmos de Diferenciação Automática (DA) (BAYDIN et al., 2018), também conhecido como Diferenciação Algorítmica. Essa técnica, DA, é mais rápida e mais precisa do que os métodos numéricos clássicos, tais como o método de diferenças finitas. Detalhes sobre diferenciação automática pode ser consultados na excelente referência (GRIEWANK; WALTHER, 2008).

O objetivo dessa dissertação é explorar o uso de redes neurais artificiais *feedforward* (redes perceptron multicamadas - PMC), no ambiente TensorFlow, na solução de problemas de valor de contorno e inicial, envolvendo equações diferenciais ordinários e parciais. Descrevemos os modelos basedos em redes neurais para soluções de EDPs e EDOs, considerando as diferentes formas de incorporar as condições de contorno e inicial na função custo. Aplicamos essa metodologia na solução de duas EDOs, no problema de Poisson (bidimensional), em dois problemas transientes (equações do calor e da onda) e em um problema singularmente pertubado unidimensional (equação de convecção-difusão). Algumas comparações, em termos de soluções gráficas e de erros medidos na norma do infinito, são realizadas com os métodos numéricos clássicos, tais como Euler, Runge-Kutta, diferenças finitas e elementos finitos. Por simplicidade, a discretização dos domínios é feita como nos métodos numéricos clássicos (não

[4] Trata-se de uma biblioteca de aprendizado de máquina extremamente flexível escrita em Python.

[5] Biblioteca de alto nível baseada no TensorFlow, com a característica de ser vantajosamente mais simples de se trabalhar do que o próprio Tensorflow;

[6] É uma biblioteca de aprendizado de máquina com foco na implementação de GPU, escrito em linguagem Lua.

utilizamos pontos aleatórios) e as condições de contorno e inicial são incorporadas na solução aproximada. A implementação computacional é desenvolvida com o sistema TensorFlow, o que proporciona algumas facilidades, tal como a utilização de funções de ativação e métodos de minimização previamente implementados, usando diferenciação automática.

O restante desse trabalho está dividido como segue:

No capítulo 2 apresentamos os principais conceitos associados às redes neurais artificiais, com a introdução do conceito de aprendizagem profunda (*deep learning*), como uma técnica poderosa em muitas aplicações;

No capítulo 3 apresentamos métodos baseados em redes neurais *perceptron* multicamadas (PMC) para resolver problemas de valor inicial e de contorno (PVIC) formado por equações diferenciais ordinárias (EDOs) e equações diferenciais parciais (EDPs);

No capítulo 4 são apresentados um conjunto de experimentos numéricos envolvendo problemas unidimensionais e bidimensionais, estacionários e transientes, onde é avaliada a performance do método RNAs;

No capítulo 5 serão apresentadas as considerações finais e recomendações para trabalhos futuros.

2 Redes Neurais Artificiais

Neste capítulo apresentamos os principais conceitos associados às redes neurais artificiais. Não pretendemos fornecer uma visão abrangente sobre o tema, mas fornecer as informações básicas para o entendimento do assunto dessa dissertação. Como leitura adicional, sugerimos as referências (HAYKIN, 2007), (JAIN; MAO; MOHIUDDIN, 1996), (SILVA; SPATTI; FLAUZINO, 2010)

2.1 Neurônios biológicos e artificiais

Antes de apresentarmos as descrições técnicas das redes neurais artificiais, vamos fornecer uma breve introdução aos neurônios biológicos, que é a fonte de inspiração para a construção de sistemas computacionais de redes neurais artificiais. Descrevemos os aspectos básicos do sistema nervoso humano, focando na forma como a informação é processada e transmitida entre redes de neurônios. Isso fornece os conceitos chaves para a descrição da formulação matemática que representa uma rede neural.

2.1.1 Neurônios biológicos

De acordo com (JAIN; MAO; MOHIUDDIN, 1996), um neurônio (ou célula nervosa) é uma célula biológica especial que processa informações, conforme Fig. 1. O neurônio é composto por um corpo celular, ou soma, e dois tipos de ramos de árvores: o axônio e os dendritos. O corpo celular tem um núcleo contendo informações sobre características hereditárias e um plasma englobando o equipamento molecular necessário para o neurônio. Um neurônio recebe sinais (impulsos) de outros neurônios através de seus dendritos (receptores) e transmite sinais gerados pelo seu corpo celular ao longo do axônio (transmissor), que eventualmente se ramifica em fios e substratos. Nos terminais desses vertentes estão as sinapses. Uma sinapse é uma estrutura elementar responsável pela comunicação entre dois neurônios ou entre um neurônio e a célula alvo. Quando o impulso chega ao terminal da sinapse, certos produtos químicos, chamados neurotransmissores, são liberados. Os neuro transmissores se difundem através do espaço sináptico, para aumentar ou inibir, dependendo do tipo de sinapse, o neurônio do receptor possui autonomia para emitir impulsos elétricos. A eficácia da sinapse pode ser ajustada pelos sinais que se deslocam através dela, por meio de atividades das quais participam. Essa dependência de fatos históricos atua como memória, que é possivelmente responsável pela memória humana. Os principais constituintes de um neurônio são

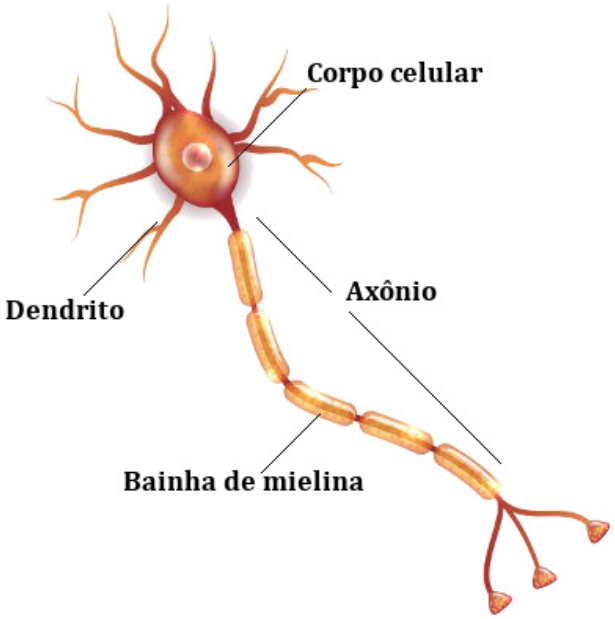


Figura 1 – Modelo de neurônio biológico (Figura extraída de (ROLO, 2019)).

- *Dentritos* - responsáveis por receber os estímulos transmitidos pelos outros neurônios;
- *Corpo celular* - responsável por coletar e combinar informações oriundas de outros neurônios;
- *Axônio* - responsável por transmitir os estímulos para outras células (constituído de uma fibra tubular que pode alcançar até alguns metros)
- Bainha de mielina - tem a função de isolar o axônio e fazer com que o estímulo percorra-o por todo o seu comprimento e chegue a outras células.

Os neurônios se comunicam através de uma sequência muito curta de pulsos, tipicamente com duração de milissegundos. A mensagem é modulada na frequência de transmissão de pulso. Isso pode variar de algumas a várias centenas de *hertz*, que é um milhão de vezes mais lento do que a velocidade de comutação mais rápida de circuitos eletrônicos. No entanto, decisões perceptivas complexas como reconhecimento facial são tipicamente feitas por humanos dentro de algumas centenas de milissegundos. Essas decisões são realizadas por uma rede de neurônios cuja velocidade operacional é de apenas alguns milissegundos.

2.1.2 Neurônios artificiais

O modelo de neurônio proposto por Mcculloch e Pitts (1943) é uma simplificação do que se sabia a respeito do neurônio biológico. McCulloch era psiquiatra e neuroanatomista

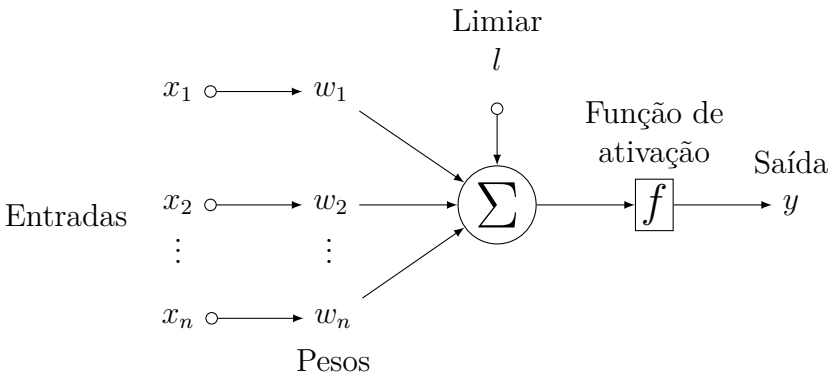


Figura 2 – Modelo de neurônio artificial

e passou cerca de 20 anos refletindo e estudando sobre a representação do sistema nervoso. Em 1942 ele convidou Pitts, que era matemático, para fazer parte das suas pesquisas; os dois foram os primeiros a descreverem um modelo artificial para um neurônio biológico, em 1943.

A descrição matemática do neurônio artificial resultou em um modelo com n terminais de entrada $x_1, x_1, ..., x_n$ (que representam os dendritos) e apenas um terminal de saída y (representando o axônio). Para emular o comportamento das sinapses, os terminais de entrada do neurônio têm pesos acoplados $w_1, w_2, ..., w_n$, cujos valores podem ser positivos ou negativos, dependendo de as sinapses correspondentes serem inibitórias ou excitatórias. O efeito de uma sinapse particular i no neurônio pós-sináptico é dado por $x_i w_i$. Os pesos determinam "em que grau" o neurônio deve considerar sinais de disparo que ocorrem naquela conexão. Atrelado a isso, é aplicada uma força externa chamada de limiar (*bias*), com o efeito de aumentar ou diminuir a entrada da função de ativação. Posteriormente ao somatório, é utilizada uma função de ativação para restringir a amplitude da saída de um neurônio, ou seja, limitar a amplitude do sinal de saída (HAYKIN, 2007). Ver Fig. 2.

2.1.3 Modelo matemático de um neurônio artificial

De acordo com Haykin (2007), um neurônio i aceita um conjunto de n entradas, $S = \{x_j | j = 1, 2, ..., n\}$. Na Fig.2, cada entrada é ponderada antes de atingir o corpo principal de um neurônio i por força de conexão ou fator de peso w_{ij} para $j = 1, 2, ..., n$. Além disso, tem um termo de limiar, l_i, que deve ser alcançado ou excedido para o neurônio i produzir um sinal de saída. Uma função $f_i : \mathbb{R} \longrightarrow \mathbb{R}$ atua no sinal produzido, chamada de função de ativação. Matematicamente, a saída do neurônio i é dada por

$$y_i = f_i\Big(\sum_{j=1}^{n} w_{ij} x_j + l_i\Big). \tag{2.1}$$

2.1.3.1 Funções de ativação

Em cada neurônio, o valor de entrada é multiplicado pelo peso correspondente e o resultado é somado com o limiar associado. Observe que essas operações são relações lineares entre as variáveis de entrada e a variável de saída do neurônio. Para que uma rede neural seja capaz de modelar relações não lineares, a saída de cada neurônio é processada por uma função, chamada função de ativação. As funções de ativação permitem as redes neurais processar saídas não lineares, especialmente nas camadas ocultas.

As funções de ativação são responsáveis também em permitir que pequenas mudanças nos pesos e limiares causem apenas uma pequena alteração na saída. Esse é o fato crucial que permitirá que uma rede neural possa aprender. Portanto, as funções de ativação decidem se um neurônio deve ser ativado ou não, ou seja, se a informação que o neurônio está recebendo é relevante para a informação fornecida ou deve ser ignorada (BRAGA; CARVALHO; LUDERMIR, 2000). Um aspecto adicional das funções de ativação é que elas devem ser computacionalmente eficientes, pois são calculadas em cada neurônio da rede para cada amostra de dados.

Existem vários tipos de funções de ativação. Destacamos aqui as seguintes funções:

- **Função de ativação logística**
  A função logística é uma função de ativação clássica, frequentemente utilizada em RNA. Existem dois tipos de funções logísticas: *sigmóide* (ou sigmoidal) e *tangente hiperbólica* (*tanh*). Essas funções são diferenciáveis e crescentes, exibindo um equilíbrio entre o comportamento linear e não linear, conforme mostrado na Fig. 3-(a) e 3-(b). A função sigmóide assume uma faixa contínua de valores de 0 a 1, caracterizada por retardar o processo de aprendizagem da rede neural (Lau; Hann Lim, 2018). A função sigmóide e sua derivada são expressas por

$$f(x) = \frac{1}{(1+e^{-ax})}, \qquad f'(x) = af(x)\Big(1-f(x)\Big), \tag{2.2}$$

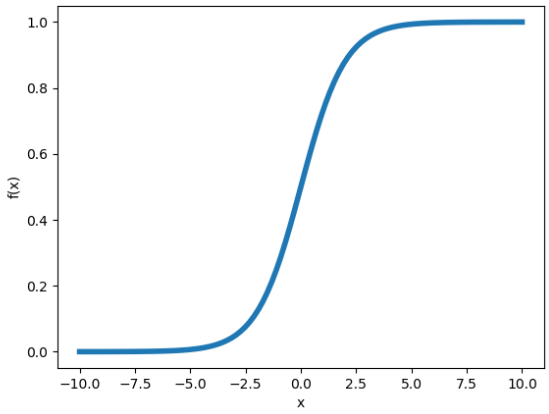
  onde $x = w_{ij}x_j + l$ e a é o parâmetro de inclinação. A função de ativação tangente hiperbólica preserva a forma da função sigmoidal, mas assume valores positivos e negativos ($f(x) \in (-1, 1)$). Essa função e sua derivada são descritas por

$$f(x) = \frac{e^{ax}-1}{e^{ax}+1}, \qquad f'(x) = a\Big(1-f(x)^2\Big). \tag{2.3}$$

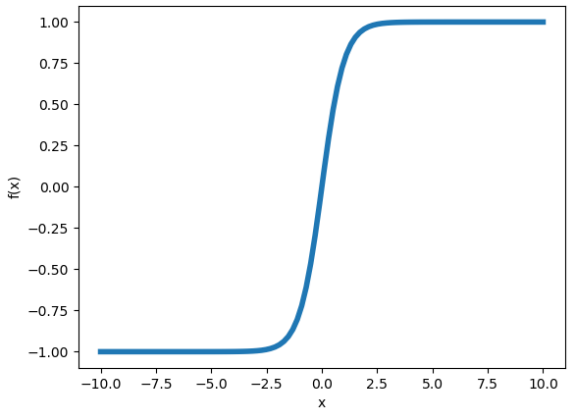
- **Função de ativação Unidade Linear Retificada** (*Rectified Linear Unit* - RELU)
  A função RELU apresentada na Fig. 3-(c) e definida em (2.4) tornou-se uma função de ativação padrão em RNA devido à sua simplicidade, pois ela diminui o tempo

computacional de treinamento. Todavia, ReLU pode sair do limite pré-definido durante o processo de treinamento da rede, logo, a taxa de aprendizagem precisa ser ajustada com cuidado, caso contrário, o neurônio pode ser inativado na função de ativação e portanto, permanecer neste processo duranto todo o processo de treinamento (GLOROT; BENGIO, 2010).

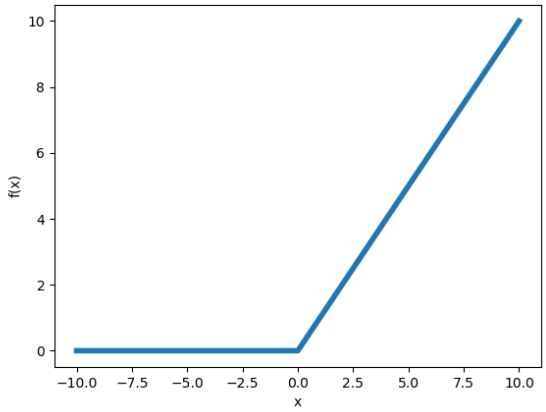
$$f(x) = max\{0, x\}. \tag{2.4}$$



(a) Função sigmóide, com $a = 1$.



(b) Função tangente hiperbólica.



(c) Função RELU.

Figura 3 – Exemplo de funções de ativação.

2.2 Arquitetura de redes neurais

A escolha do tipo de arquitetura é muito importante, uma vez que esta relaciona-se diretamente ao tipo de problema a ser tratado pela rede (BRAGA; CARVALHO; LUDERMIR,

2000). Nesta seção são apresentadas as principais arquiteturas (estruturas) de rede utilizadas bem como as características que as diferenciam. A Fig. 4 apresenta alguns exemplos de redes neurais artificiais.

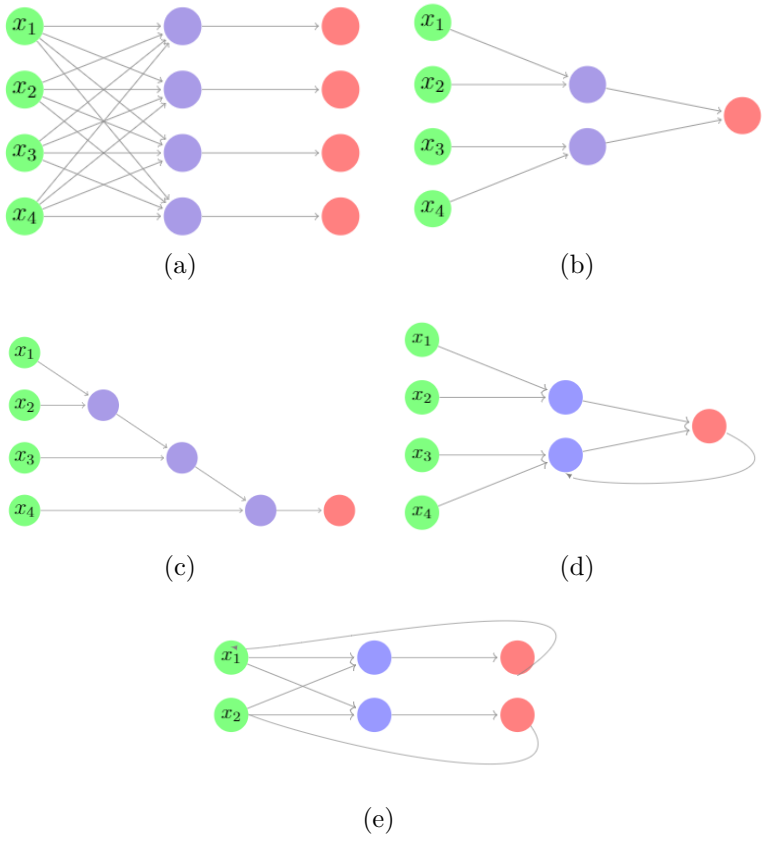


Figura 4 – Exemplos de arquiteturas de Redes Neurais Artificiais

As arquiteturas de redes neurais artificiais podem ser classificadas de acordo com o número de camadas,

- Redes de camada única - existe apenas um nó entre qualquer entrada e toda saída (figuras 4-(a) e 4-(e));
- Redes de múltiplas camadas - existem mais do que uma camada entre alguma entrada e alguma saída (figuras 4-(b), 4-(c) e 4-(d)).

Ainda sobre a arquitetura, estas diferenciam-se de acordo com os tipos de conexões,

- *Feedfoward*, ou acíclica - a saída de um neurônio da i-ésima camada da rede não pode ser usada como entrada de nós em camadas de índice menor ou igual a i (figuras 4-(a), 4-(b) e 4-(c));

- *Feedback*, ou cíclica - a saída de um neurônio da i-ésima camada da rede é usada como entrada de nós em camadas de índice menor ou igual a i (figuras 4-(d) e 4-(e)).

Por fim, as RNAs podem ser classificadas conforme sua conectividade,

- Rede fracamente (ou parcialmente) conectada (Fig. 4-(d));
- Rede completamente conectada (Fig. 4-(e)).

2.3 Processos de treinamento e aprendizagem

O treinamento de uma rede neural artificial é o procedimento pelo qual os valores dos pesos e limiares são determinados de modo que o problema que a rede está modelando seja resolvido com precisão (GARDNER; DORLING, 1998). Assim, o aprendizado de uma RNA ocorre com o ajuste ótimo dos parâmetros livres da rede.

Dado um conjunto de de entrada, o ajuste ótimo dos pesos sinápticos de uma rede permite que a rede apresente a saída desejada, com um nível de erro aceitável. De acordo com (SATHYA; ABRAHAM, 2013), os paradigmas de aprendizagem da RNA podem ser classificados em aprendizagem supervisionada, não supervisionada e por reforço. O modelo de aprendizagem supervisionada pressupõe a disponibilidade de um professor ou supervisor que classifica os exemplos de treinamento em classes e utiliza as informações sobre os membros da classe de cada instância de treinamento, enquanto o modelo de aprendizagem não supervisionada identifica as informações da classe padrão heuristicamente. A aprendizagem por reforço é obtida por tentativa e erro, através de interações com seu ambiente (atribuição de recompensa/penalidade).

2.4 Modelos de redes neurais artificiais

Os Modelos de redes neurais artificias possuem inúmeras aplicações em diversas áreas da ciência e engenharia, como astronomia, biologia, finanças, geologia, etc. Embora algumas arquiteturas básicas de RNAs tenham sido introduzidas há muito tempo, como as arquiteturas do tipo $feedforward$, elas continuam sendo muito importantes em aplicações de grande relevância, mais recentemente na forma de aprendizagem profunda (*deep learning*) (TINO; BENUSKOVA; SPERDUTI, 2015). Nesta seção serão apresentados alguns dos principais modelos de RNAs.

2.4.1 Rede *Perceptron*

O *Percepton*, introduzido por Rosenblatt (1957), é considerada a configuração mais simples de uma RNA, devido sua estrutura apresentar apenas sinais de entrada, representativos do problema, uma camada neural, contendo somente um neurônio artificial e somente uma saída, remetendo-se à arquiteturas *feedforward*.

2.4.2 Rede Adaline

A rede Adaline foi desenvolvida em 1960 por Widrow Hoff (WIDROW, 1960), sendo descrita de forma semelhante a rede *Perceptron*, porém com uma nova regra de aprendizado supervisionado, denominada regra delta. Assim como no Perceptron, a rede Adaline possui uma camada de entrada com n unidades e uma camada de saída com apenas um neurônio.

A rede Adaline possui arquitetura *feedforward*, pois o fluxo de informação é realizado partindo-se das entradas em direção a saída da rede.

2.5 Redes *Perceptrons* de múltiplas camadas

Uma rede *perceptron* de múltiplas camadas (PMC) pode ter uma ou mais camadas ocultas e, por fim, uma camada de saída. Perceptrons com múltiplas camadas são descritos como sendo totalmente conectados, com cada nó conectado a cada nó na camada anterior e a seguinte (GARDNER; DORLING, 1998).

O número de camadas ocultas em um PMC e o número de nós em cada camada podem variar para um determinado problema. Contudo, o aumento de nós implica em uma maior sensibilidade ao problema a ser resolvido, como o risco de sobreajuste dos pesos (MURTAGH, 1991). De acordo com Cybenko (1988), uma rede neural com apenas uma camada oculta é capaz de sempre aproximar uma função contínua.

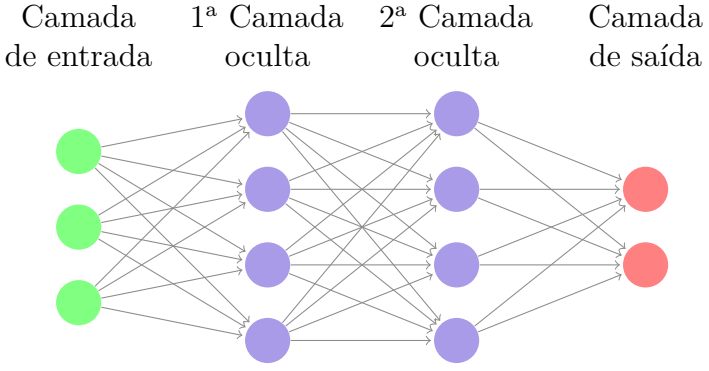


Figura 5 – Representação de uma rede *perceptron* multicamadas (duas camadas ocultas).

A rede PMC pertence à arquitetura *feedfoward* de camadas múltiplas, cujo treinamento é realizado de forma supervisionada. Conforme observado na Fig. 5, o fluxo de informações na estrutura da rede se inicia na camada de entrada, percorre em seguida as camadas intermediárias (ocultas), sendo então finalizado na camada neural de saída. Desta maneira, não existe a possibilidade de realimentação de valores.

Um perceptron múltiplas camadas possui três características distintas (HAYKIN, 2007):

1. O modelo de cada neurônio inclui uma função de ativação não linear, por exemplo, a função sigmoidal.

2. A rede contém uma ou mais camadas ocultas de neurônios. Os neurônios pertencentes as camadas ocultas capacitam a rede a aprender tarefas complexas extraindo progressivamente as características mais significativas dos padrões (vetores) de entrada.

3. A rede exibe um alto grau de conectividade, determinado pelas sinapses dos neurônios. Uma modificação na conectividade da rede requer uma mudança na população das conexões sinápticas ou de seus pesos.

2.5.1 Princípio de funcionamento do Perceptron Multicamadas

Conforme Fig. 5, observa-se que cada uma das entradas da rede, representando os sinais advindos de determinada aplicação, será propagada em direção à camada neural de saída. Neste caso, as saídas dos neurônios da primeira camada neural oculta serão as próprias saídas daqueles neurônios pertencentes à segunda camada neural escondida. Para a situação da rede ilustrada na Fig. 5, as saídas dos neurônios da segunda camada neural escondida serão as respectivas entradas dos neurônios pertencentes à sua camada neural de saída.

Assim, a propagação dos sinais de entradas da rede PMC, independentemente da quantidade de camadas intermediárias, é sempre realizada num único sentido, ou seja, da camada de entrada em direção à camada neural de saída.

Diferentemente da rede *Perceptron* e da rede *Adaline*, além da presença de camadas ocultas, a camada neural de saída do PMC pode ser composta por diversos neurônios: cada um destes neurônios representaria uma das saídas do processo a ser mapeado. As camadas intermediárias, por sua vez, extraem a maioria das informações referentes ao seu comportamento e as codificam por meio dos pesos sinápticos e limiares de seus neurônios.

A especificação da configuração topológica de uma rede PMC, tais como quantidade de camadas intermediárias e seus respectivos números de neurônios, depende de diversos fatores,

dentre os quais: a classe de problema a ser tratada pelo PMC, a disposição espacial das amostras de treinamento e os valores iniciais atribuídos tanto aos parâmetros de treinamento quanto às matrizes de pesos. O algoritmo clássico de aprendizagem utilizado durante o processo de treinamento de redes PMC é conhecido como *backpropagation* ou algoritmo de retropropagação do erro (HAYKIN, 2007).

2.5.2 Processo de treinamento da rede *perceptron* multicamadas

No processo de treinamento de redes PMC é aplicado um algoritmo de aprendizado denominado *backpropagation* ou algoritmo de retropropagação do erro, conhecido também como regra Delta generalizada (BRAGA; CARVALHO; LUDERMIR, 2000). A ilustração das fases é mostrada na Fig. 6, na qual é utilizada uma configuração de PMC constituída de duas camadas ocultas composta de n sinais na camada de entrada, e ainda, com n_1 na primeira camada oculta, n_2 na segunda camada oculta e n_3 sinais associados na camada neural de saída (última camada neural).

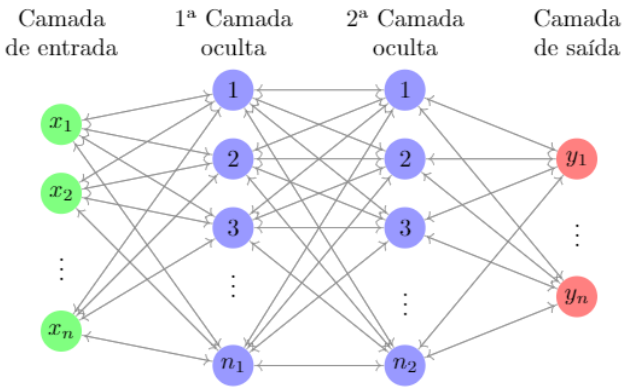


Figura 6 – Apresentação das duas fases de treinamento do Perceptron multicamadas.

O treinamento de uma rede PMC é geralmente constituído de duas fases bem especificas. A primeira fase a é denominada de "propagação adiante" (*forward*), onde visa-se obter as respostas da rede levando em consideração, apenas valores atuais dos pesos sinápticos e limiares de seus neurônios, que serão inalterados durante cada execução desta fase. Nesta fase os sinais de entrada $x_1, x_2, ..., x_n$ de uma amostra do conjunto de treinamento são inseridos nas entradas da rede e são propagados camada a camada até a produção das respectivas saídas. Após isso as saídas da rede são comparadas com as respectivas respostas desejadas que estejam disponíveis.

Considerando uma rede PMC que contenha n neurônios em sua camada de saída, os respectivos n desvios (erros) diferença entre as respostas desejadas e aquelas obtidas pela rede são então calculados, os quais posteriormente serão utilizados para ajustar os pesos e limiares

de todos os seus neurônios. Em seguida aplica-se, a segunda fase do método *backpropagation* denominada de "propagação reversa" (*backward*). No decorrer desta fase, diferentemente da fase anterior, são executadas as alterações (ajuste) dos pesos sinápticos e limiares de todos os neurônios da rede.

As sucessivas aplicações das fases *forward* e *backward* fazem com que os pesos sinápticos e limiares dos neurônios se ajustem automaticamente em cada interação, implicando na gradativa diminuição da soma dos erros produzidos pelas respostas da rede frente aquelas desejadas.

Para um melhor entendimento da funcionalidade envolvida no algoritmo *backpropagation*, serão primeiramente definidas diversas variáveis e parâmetros auxiliares. Baseando-se na topologia de PMC apresentada na Fig. 6, são apresentadas, vide Fig. 7, um conjunto de variáveis que direcionam a derivação do algoritmo.

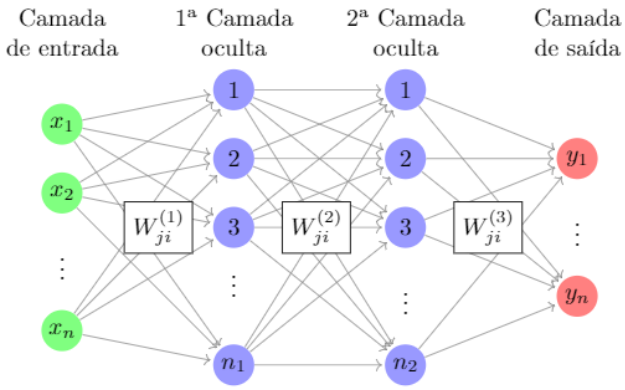


Figura 7 – Configuração do neurônio utilizado na derivação do algoritmo *backpropagation*.

Será considerado cada um dos neurônios $\{j\}$ pertencentes a uma das camadas $\{L\}$ como mostra a Fig. 5 como um perceptron simples, ilustrado na Figura 7, onde $g(.)$ representa a função de ativação, sendo a mesma diferenciável em todo seu domínio, a exemplo da tangente hiperbólica.

A partir das figuras 5 e 7 são utilizadas as seguintes terminologias para os seus parâmetros constituintes:

- W_{ji}^{L} representa os elementos da matriz W^L de pesos cujos elementos denotam o valor do peso sináptico conectando o j-ésimo neurônio da camada L ao i-ésimo neurônio da camada $L-1$. A camada de entrada é indicada com $L=0$. Para a topologia ilustrada na Fig. 7 tem-se:
  - W_{ji}^{3} é o peso sináptico conectando o j-ésimo neurônio da camada de saída ao i-ésimo neurônio da camada 2;

  - W^2_{ji} é o peso sináptico conectando o j-ésimo neurônio da camada escondida 2 ao i-ésimo neurônio da camada 1;
  - W^1_{ji} é o peso sináptico conectando o j-ésimo neurônio da camada escondida 1 ao i-ésimo neurônio da camada de entrada.

- I^L_j denota a entrada ponderada em relação ao j-ésimo neurônio da camada L. Por exemplo,

$$I^1_j = \sum_{i=0}^{n} W^1_{ji} x_i = W^1_{j,0} x_0 + W^1_{j,1} x_0 + \cdots + W^1_{j,n} x_n; \tag{2.5}$$

$$I^2_j = \sum_{i=0}^{n1} W^2_{ji} Y^1_i = W^2_{j,0} Y^1_0 + W^2_{j,1} Y^1_0 + \cdots + W^2_{j,n_1} Y^1_{n_1}; \tag{2.6}$$

$$I^3_j = \sum_{i=0}^{n2} W^3_{ji} Y^2_i = W^3_{j,0}.Y^2_0 + W^3_{j,1} Y^2_0 + \cdots + W^3_{j,n_2} Y^2_{n_2}. \tag{2.7}$$

- Y^L_j denota a saída do j-ésimo neurônio em relação à camada L, os quais são definidos por

$$Y^{(1)}_j = g(I^{(1)}_j); \tag{2.8}$$

$$Y^{(2)}_j = g(I^{(2)}_j); \tag{2.9}$$

$$Y^{(3)}_j = g(I^{(3)}_j). \tag{2.10}$$

Todo o processo de treinamento do PMC ocorre em função do erro, dado pela equação

$$e^{(k)}_j = d_j(k) - y_j(k), \tag{2.11}$$

onde $y_j(k)$ é a resposta calculada para o neurônio j e $d_j(k)$ é a resposta desejada para o neurônio j.

De acordo com a Fig. 7, na k-ésima amostra de treinamento, a função erro quadrático médio mede o desempenho local associado aos resultados produzidos pelos nerônios de saída frente a referida amostra, logo

$$E(k) = \frac{1}{2} \sum_{j=1}^{n3} (d_j(k) - y^{(3)}_j(k))^2, \tag{2.12}$$

onde $y_j^3(k)$ é o valor produzido pelo j-ésimo neurônio de saída da rede considerando-se a k-ésima amostra de treinamento, enquanto que $d_j(k)$ é o seu respectivo valor desejado.

Assumindo um conjunto de treinamento composto por p amostra, o erro quadrático médio é escrito como

$$E(M) = \frac{1}{p}\sum_{k=1}^{p} E(k). \tag{2.13}$$

O método utilizado para o ajuste dos pesos e limiares é baseado no gradiente (também chamado de método da descida mais íngreme) da função erro quadrático dada em (2.12).

2.5.3 Ajuste dos pesos sinápticos da camada de saída

O objetivo é determinar o gradiente $\nabla E^{(3)}$, para aplicar uma correção $\Delta W_{ij}^{(3)}$ na matriz de pesos $W^{(3)}$ com o intuíto de minimizar o erro entre a saída produzida pela rede e sua respectiva saída desejada (conforme apresentado em (SILVA; SPATTI; FLAUZINO, 2010). De acordo com a regra da cadeia, o gradiente pode ser expresso como

$$\nabla E^{(3)} = \frac{\partial E}{\partial W_{ji}^3} = \frac{\partial E}{\partial Y_j^3}\frac{\partial Y_j^3}{\partial I_j^3}\frac{\partial I_j^3}{\partial W_j^3}. \tag{2.14}$$

Por intermédio das definições anteriores temos que

$$\frac{\partial I_j^3}{\partial W_{ji}^3} = Y_i^2, \tag{2.15}$$

$$\frac{\partial Y_j^3}{\partial I_j^3} = g'(I_j^3) \tag{2.16}$$

e

$$\frac{\partial E_j^3}{\partial Y_j^3} = -(d_j - Y_j^3), \tag{2.17}$$

onde $g'(\cdot)$ denota a derivada de primeira ordem da função de ativação considerada. Substituindo os resultados das equações (2.15), (2.16) e (2.17) em 2.14, obtém-se

$$\frac{\partial E}{\partial W_{ji}^3} = -(d_j - Y_j^3)g'(I_j^3)Y_i^2. \tag{2.18}$$

Logo, o ajuste da matriz de pesos W^1 deve ser efetuado em direção oposta ao gradiente a fim de minimizar o erro (método da descida mais íngreme), ou seja

$$\Delta W_{ji}^3 = -\eta\frac{\partial E}{\partial W_{ji}^3}\Delta W_{ji}^3 = -\eta\delta_j^3 Y_i^2. \tag{2.19}$$

2.6 Aprendizagem profunda (*Deep learning*)

Os enormes avanços científicos e tecnológicos obtidos através da computação e o crescimento exponencial na coleta e disponibilidade de dados nas últimas décadas coincidiram com um aumento do interesse no campo de aprendizagem de máquinas (*machine learning*). As aplicações são amplas, por exemplo, reconhecimento de imagens e fala, diagnósticos médicos, etc.

A tecnologia de aprendizagem de máquina impulsiona muitos aspectos da sociedade moderna, como pesquisa na web, filtragem de conteúdo em redes sociais, recomendações em sites de comércio eletrônico, e está mais presente em produtos de consumo, como smartphones e câmeras (LECUN; BENGIO; HINTON, 2015). O aprendizado de máquina é o núcleo da área de inteligência artificial. É um assunto multidisciplinar e interdisciplinar, envolvendo teoria da probabilidade, estatística, teoria da aproximação, análise convexa, teoria da complexidade de algoritmos bem como outras disciplinas. Fundamenta-se em como os computadores simulam ou implementam o comportamento de aprendizagem humana para adquirir novos conhecimentos ou habilidades e reorganizar as estruturas de conhecimento existentes para melhorar seu próprio desempenho (WANG; FAN; WANG, 2020). De um modo geral, o aprendizado de máquina consiste em aprender regras a partir de um grande número de dados por meio de algoritmos relacionados e fazer conjecturas ou julgamentos sobre novos dados de amostra e, equivalente ao aprender em seres humanos.

As redes neurais artificiais são modelos de aprendizagem de máquinas que têm recebido muita atenção nos últimos anos devido ao seu sucesso em uma série de aplicações diferentes (LECUN; BENGIO; HINTON, 2015). O termo *deep learning* se refere ao uso de redes neurais artificiais com muitas camadas ocultas em problemas de aprendizagem de máquina.

2.7 Aprendizagem profunda na solução de equações diferenciais

Resolver equações diferenciais numericamente, principalmente as equações diferenciais parciais, é uma tarefa muito utilizada em aplicações científicas e no campo das engenharias. Existem vários métodos numéricos para resolver o moldeo matemático resultante, tais como o método de diferenças finitas, método de elementos finitos e método de volumes finitos, etc. Embora estes métodos sejam eficientes, eles possuem algumas limitações, como soluções discretas (definidas em pontos de uma malha) ou com diferenciabilidade limitada. Nos últimos quatro anos surgiram vários trabalhos na literatura abordando o uso de *deep learning* na solução de equações diferenciais parciais. Em particular, Sirignano e Spiliopoulos (2018) apresentam o método livre de malha, *Deep Galerkin Method* (DGM), para resolver EDPs definidas

em domínios com alta dimensão, utilizando técnicas de *deep learning*. Técnicas numéricas clássicas, como métodos de diferenças finitas e elementos finitos, não são efetivas na solução de problemas que dependem de muitas variáveis independentes, devido ao grande número de pontos nodais nas malhas que discretizam o domínio. Em geral, os problemas modelados no contexto da mecânica do contínuo podem depender de quatro variáves independentes (três espaciais e uma temporal), mas existem problemas que podem ser descritos em termos de centenas de variáveis independentes, como àqueles que aparecem na área de finanças. Métodos baseados em *deep learning* têm se mostrado eficientes na solução desses problemas.

2.7.1 Teorema da aproximação universal

RNAs são conhecidas como uma ferramenta poderosa na aproximação de funções matemáticas. Esse fato é verdadeiro graças ao *teorema da aproximação universal* (CYBENKO, 1989; HORNIK; STINCHCOMBE; WHITE, 1989). Esse teorema afirma que RNAs *feedforward* podem aproximar qualquer função contínua em um conjunto compacto[1], com uma precisão arbitrária. Além disso, a rede neural precisa ter apenas uma única camada oculta, com um número finito de neurônios, para obter esse resultado.

Na verdade, existem vários teoremas de aproximação universal, cada um com suas particularidades. Todos eles estabelecem que, dados os pesos apropriados, RNAs podem representar uma ampla variedade de funções matemáticas interessantes. Normalmente, eles não fornecem um processo de construção para os pesos, apenas afirmam que tal construção é possível. Uma das primeiras versões desse teorema (talvez, a mais famosa) foi provada por Cybenko (1989), para funções de ativação do tipo sigmóide. Segundo Hornik (1991), não é a escolha da função de ativação que dá às redes neurais o potencial de serem aproximadores universais, mas sim a própria arquitetura *feedforward* multicamadas. Leshno et al. (1993) mostrou que uma rede *feedforward* multicamada com uma função de ativação contínua por partes e limitada localmente pode aproximar qualquer função contínua com qualquer grau de precisão se e somente se a função de ativação não for um polinômio. Um resultado similar foi mostrado em (PINKUS, 1999). Vários resultados e extensões do teorema de aproximação universal são descritos na literatura. O autor interessado, além dos artigos citados nessa seção, pode também pesquisar as referências (LU et al., 2017; HANIN; SELLKE, 2018; KIDGER; LYONS, 2020; PARK et al., 2020; KRATSIOS, 2021).

O teorema da aproximação universal, de Cybenko (1989), para redes MLP é descrito a seguir. Para isso, vamos considerar uma rede neural artificial com três camadas (camadas de entrada, oculta e de saída), de forma que a camada de entrada contém m nós, com valores

[1] Em dimensão finita, um conjunto compacto é um conjunto fechado e limitado.

$x_1, \cdots, x_m$, a (única) camada oculta contém m_H neurônios, onde seu i -ésimo neurônio possui pesos associado $w_{i1}, ..., w_{im}$ e limiar l_i, enquanto que a camada de saída possui um único neurônio que fornece uma combinação linear das saídas das unidades da camada oculta, com $v_1, \cdots, v_{m_H}$ definindo os coeficientes dessa combinação linear (ou pesos entre a camada oculta e a camada de saída). A saída dessa rede neural é dada por

$$G(\boldsymbol{x}) = G(x_1, \cdots, x_m) = \sum_{i=1}^{m_H} v_i \phi\Big(\sum_{j=1}^{m} w_{ij} x_j - l_i \Big), \tag{2.20}$$

onde $\phi : \mathbb{R} \longrightarrow \mathbb{R}$ é a função de ativação.

Teorema 2.7.1 *Sejam $\phi : \mathbb{R} \longrightarrow \mathbb{R}$ uma função (de ativação) contínua, limitada e monotonicamente crescente, $I_m = (0,1)^m$ um hipercubo unitário de dimensão m e $\mathcal{C}(I_m)$ o conjunto de todas as funções contínuas definidas em I_m. Dados uma função $f \in \mathcal{C}(I_m)$ e $\epsilon > 0$, arbitrário, existem inteiros positivos m e m_H, valores reais v_i e w_{ij}, $i = 1, \cdots, m_H$, $j = 1, \cdots, m$, tais que*

$$\Big| f(x_1, \cdots x_m) - G(x_1, \cdots, x_m) \Big| < \epsilon, \tag{2.21}$$

para todo $(x_1, \cdots x_m) \in I_m$, onde a função G é dada por (2.20).

Demonstração 2.8 *Ver (CYBENKO, 1989).*

3 Redes *perceptron* multicamadas aplicadas na solução de equações diferenciais

Neste capítulo apresentamos métodos baseados em redes neurais *perceptron* multicamadas (PMC) para resolver Problemas de Valor de Inicial e de Contorno (PVIC) formado por Equações Diferenciais Ordinárias (EDOs) e Equações Diferenciais Parciais (EDPs).

O processo de solução de equações diferenciais via redes neurais artificiais consiste em transformar o modelo original em um funcional quadrático associado ao resíduo da equação e em seguida, minimizar esse funcional usando técnicas de otimização irrestrita no contexto de uma rede neural escolhida a priori. A descrição desse funcional quadrático envolve o operador diferencial que representa o modelo matemático, as condições de contorno e iniciais, os pontos nodais do particionamento do domínio e um vetor incógnita que descreve os parâmetros (pesos e limiares) ótimos da rede neural.

Vamos considerar um modelo matemático formado por uma equação diferencial parabólica com condições de contorno de Dirichlet. O problema consiste em determinar a função escalar $u : \Omega \times (0, T_f] \longrightarrow \mathbb{R}$ tal que

$$\begin{aligned} \frac{\partial u(\boldsymbol{x},t)}{\partial t} + \mathcal{L}u(\boldsymbol{x},t) &= f(\boldsymbol{x},t), \quad \forall (\boldsymbol{x},t) \in \Omega \times (0,T_f], && (3.1)\\ u(\boldsymbol{x},t) &= g(\boldsymbol{x},t), \quad \forall (\boldsymbol{x},t) \in \Gamma \times (0,T_f], && (3.2)\\ u(\boldsymbol{x},0) &= u_0(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega, && (3.3) \end{aligned}$$

onde $\Omega \subset \mathbb{R}^d$ é um domínio aberto e limitado com fronteira Γ, $(0, T_f]$ é o intervalo temporal, com $T_f > 0$, $\boldsymbol{x} = (x_1, \cdots, x_d) \in \Omega$, f e g são funções dadas (f representa o termo de fonte ou sorvedouro e g a condição de contorno de Dirichlet), u_0 é a condição inicial dada e $\mathcal{L}(\cdot)$ é um operador diferencial de segunda ordem (elíptico). Eq. (3.1) representa a equação diferencial dependente do tempo (transiente) a ser resolvida, com condições de contorno de Dirichlet dadas por (3.2) e condição inicial (3.3). Se não há variação temporal, o problema é dito ser estacionário ou permanente. Neste caso, o problema (3.1)-(3.3) resulta em determinar a função escalar $u : \Omega \longrightarrow \mathbb{R}$ tal que

$$\begin{aligned} \mathcal{L}u(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega, && (3.4)\\ u(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Gamma. && (3.5) \end{aligned}$$

Como exemplo do operador diferencial $\mathcal{L}(\cdot)$, destacamos,

a)

$$\mathcal{L}(u) = -\Delta u = -\Big(\sum_{i=1}^{d} \frac{\partial^2 u}{\partial x_i^2}\Big).$$

Neste caso, o modelo matemático (3.1)-(3.3) pode descrever a variação do campo de temperatura u no espaço e no tempo (equação do calor) ou a variação da concentração u de uma certa subtância no espaço e no tempo, considerando somente processos difusivos e temporal (equação de tranporte difusivo)(QUARTERONI, 2009).

b)

$$\mathcal{L}(u) = -\epsilon\Delta u + \boldsymbol{\beta}\cdot\boldsymbol{\nabla} u + cu,$$

onde $\epsilon > 0$, $c \geq 0$, $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_d)$ e $\boldsymbol{\nabla} u = (\frac{\partial u}{\partial x_1}, \cdots, \frac{\partial u}{\partial x_d})$. Neste caso, o modelo matemático (3.1)-(3.3) é conhecido como problema de difusão-convecção-reação, com ϵ sendo o coeficiente de difusão, $\boldsymbol{\beta}$ o campo de velocidade e c o coeficiente de reação (QUARTERONI, 2009).

Se uma função $u : \Omega \times (0, T_f] \longrightarrow \mathbb{R}$ satifaz o problema (3.1) em todos os pontos de Ω e $(0, T_f]$, além das condições de contorno e inicial, dizemos que u é a solução exata (clássica) do problema (3.1)-(3.3). Obter soluções exatas para equações diferenciais nem sempre é possível, por isso o desenvolvimento de métodos numéricos para resolver PVICs é uma área de pesquisa bastante ativa. Usando ferramentas numéricas com estruturas de dados eficientes é possível resolver de forma aproximada uma variedade considerável de problemas modelados por equações diferenciais ordinárias e parciais. O objetivo de um método numérico é fornecer uma solução aproximada u_h de u tal que o erro absoluto

$$e = \|u - u_h\|_V$$

seja o menor possível, onde $\|\cdot\|_V$ é uma norma específica associada a um espaço de funções V, tal que $u \in V$. Como a solução exata dos problemas de interesse nem sempre é conhecida, a avaliação do erro absoluto é impossível. Um outra forma de avaliar a qualidade da solução aproximada pode ser através do resíduo da equação diferencial. Se u_h é uma solução aproximada para u, então o resíduo é dado por

$$R(u_h) = \frac{\partial u_h}{\partial t} + \mathcal{L}(u_h) - f. \tag{3.6}$$

Note que $R(u_h) \approx 0$ implica em $\frac{\partial u_h}{\partial t} + \mathcal{L}(u_h) \approx f$ e u_h é uma boa aproximação para o problema (supondo que u_h satisfaça as condição inicial e de contorno).

Dentre os métodos numéricos utilizados para resolver PVICs, destacam os métodos de Elementos Finitos, Diferenças Finitas e Volumes Finitos (QUARTERONI, 2009). Uma

característica padrão desses métodos é a necessidade do uso de uma malha que representa uma discretização do domínio $\Omega \times [0, T_f]$. Em problemas estacionários, cada ponto da discretização de Ω associa a uma equação algébrica de forma que o conjunto de pontos definido pela malha resulta em um sistema de equações (que pode ser não linear se o problema for não linear) a ser resolvida. A aplicabilidade dessas técnicas a problemas de dimensão maior ou igual a 3, como àqueles que aparecem na área de finanças (cuja dimensão pode atingir a ordem de centenas), não é recomendada, devido ao alto desempenho computacional (SIRIGNANO; SPILIOPOULOS, 2018).

Métodos baseados em redes neurais para resolver o problema (3.1)-(3.3) busca obter uma função u_{RN} que aproxima a solução exata u, através da minimização do quadrado do resíduo (3.6), em termos dos parâmetros de ajuste (pesos e limiares) da rede neural. Uma característica importante da solução aproximada u_{RN} é que ela pode ser uma função infinitamente diferenciável, dependendo da função de ativação utilizada, situação que não ocorre nas soluções obtidas pelos métodos de elementos finitos, diferenças finitas e volumes finitos.

A solução aproximada u_{RN} é descrita em função dos dados de entrada, dos parâmetros de ajuste (pesos e limiares) e da função saída da rede neural, isto é,

$$u_{RN} = u_{RN}(\boldsymbol{x}, t, N(\boldsymbol{x}, t, \boldsymbol{p})) = u_{RN}(\boldsymbol{x}, t, \boldsymbol{p}),$$

onde N é a função saída da rede neural e $\boldsymbol{p}$ é um vetor cujas componentes são os parâmetros de ajuste da rede, que são os pesos e limiares. A função N depende da arquitetura da rede neural e será apresentada na Seção 3.3. A minimização do resíduo (3.6) pode ser obtido através da minimização de um funcional quadrático, definido por

$$J_R(\boldsymbol{p}) = \|R(u_{RN}(\boldsymbol{x}, t, \boldsymbol{p}))\|^2_{(\Omega\times[0,T_f])}, \tag{3.7}$$

em uma malha fixa de pontos (construída em $\Omega \times [0, T_f]$). Esse funcional varia em função dos parâmetros de ajuste da rede neural, descritos pelo vetor $\boldsymbol{p}$. Vale ressaltar que o mínimo de (3.7) é obtido de forma que u_{RN} satisfaça as condições inicial e de contorno, isto é,

$$u_{RN}(\boldsymbol{x}, t, \boldsymbol{p}) = g(\boldsymbol{x}, t), \quad \forall (\boldsymbol{x}, t) \in \Gamma \times (0, T_f], \tag{3.8}$$

$$u_{RN}(\boldsymbol{x}, 0, \boldsymbol{p}) = u_0(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega. \tag{3.9}$$

A norma $\|\cdot\|_{(\Omega\times[0,T_f])}$ deve ser escolhida de forma apropriada. Por exemplo, podemos utilizar a norma usual do espaço das funções de quadrado integráveis (no sentido de Lebesgue), $L^2(\Omega \times [0, T_f])$, definida por

$$\|w(\boldsymbol{y})\|^2_{L^2(\Omega\times[0,T_f])} = \int_{(\Omega\times[0,T_f])} |w(\boldsymbol{y})|^2 d\boldsymbol{y}.$$

O funcional (3.7), no contexto de redes neurais, é conhecido com vários nomes, como função custo, função erro, função objetivo, etc. Neste trabalho, chamaremos esse funcional de *funcional resíduo*.

O problema de minimizar (3.7) sujeito às restrições (3.8) e (3.9) é um problema de otimização/minimização restrita. O objetivo é determinar o vetor de parâmetros $\boldsymbol{p}$ que minimiza o funcional J_R, sujeito às restrições de igualdade dadas pelas condições inicial e de contorno. O vetor $\boldsymbol{p}$ ótimo fornecerá a solução aproximada u_{RN}.

De acordo com Rudd (2013), uma das principais dificuldades em resolver modelos matemáticos baseado em equações diferenciais através de redes neurais é a satisfação das condições inicial e de contorno. Essas condições são restrições de igualdade sobre um domínio contínuo. Existem na literatura duas formas principais de tratamento das condições inicial e de contorno,

1. Adicionar termos que penalizam as condições inicial e de contorno no funcional resíduo;
2. Usar soluções aproximadas que satisfazem automaticamentes as condições inicial e de contorno.

Ambas estratégias transformam o problema de minimização restrita em um problema de minimização irrestrita. A seções 3.1 e 3.2 descrevem em detalhes essas duas estratégias.

A minimização do funcional resíduo é realizada em uma malha de pontos do domínio $(\Omega \times [0, T_f]) \cup \Gamma$. Considere discretizações do domínio Ω,

$$\mathcal{P}_\Omega = \{\boldsymbol{x}_i \in \Omega; i = 1, \cdots, n_\Omega\},$$

do domínio temporal $[0, T_f]$,

$$\mathcal{P}_t = \{t_j \in [0, T_f]; j = 1, \cdots, n_t\}$$

e da fronteira Γ,

$$\mathcal{P}_\Gamma = \{\boldsymbol{x}_k \in \Gamma; k = 1, \cdots, n_\Gamma\}.$$

Alguns trabalhos consideram uma malha fixa de pontos obedecedo uma uniformidade, outros consideram simplesmente um agrupamento (ou vários agrupamentos) aleatório de pontos, de forma que o método seja considerado não dependente de malha.

O problema de minimização contínua é então transformado em um problema de minimização discreta. Para isso, o problema contínuo é transformado no seguinte sistema de equações,

$$\frac{\partial u_{RN}(\boldsymbol{x}_i, t_j, \boldsymbol{p})}{\partial t} + \mathcal{L}(u_{RN}(\boldsymbol{x}_i, t_j, \boldsymbol{p})) = f(\boldsymbol{x}_i, t_j), \quad \forall (\boldsymbol{x}_i, t_j) \in \mathcal{P}_\Omega \times \mathcal{P}_t, \quad i = 1, \cdots, n_\Omega, \tag{3.10}$$

$j = 1, \cdots, n_t$ e

$$u_{RN}(\boldsymbol{x}_k, t_j, \boldsymbol{p}) = g(\boldsymbol{x}_k, t_j), \quad \forall (\boldsymbol{x}_k, t_j) \in \mathcal{P}_\Gamma \times \mathcal{P}_t, \ k = 1, \cdots, n_\Gamma, j = 1, \cdots, n_t, \tag{3.11}$$

$$u_{RN}(\boldsymbol{x}_i, 0, \boldsymbol{p}) = u_0(\boldsymbol{x}_i), \quad \forall \boldsymbol{x}_i \in \mathcal{P}_\Omega, \quad i = 1, \cdots, n_\Omega. \tag{3.12}$$

Define-se o resíduo (discreto) associado a (3.10) em função de $\boldsymbol{p}$ no ponto $(\boldsymbol{x}_i, t_j)$, $i = 1, \cdots, n_\Omega$, $j = 1, \cdots, n_t$, por

$$R_{ij}(\boldsymbol{p}) = \frac{\partial u_{RN}(\boldsymbol{x}_i, t_j, \boldsymbol{p})}{\partial t} + \mathcal{L}(u_{RN}(\boldsymbol{x}_i, t_j, \boldsymbol{p})) - f(\boldsymbol{x}_i, t_j)$$

e o funcional resíduo quadrático discreto,

$$\begin{aligned} J_R(\boldsymbol{p}) &= \sum_{\boldsymbol{x}_i \in \mathcal{P}_\Omega} \sum_{t_j \in \mathcal{P}_t} (R_{ij}(\boldsymbol{p}))^2 \\ &= \sum_{\boldsymbol{x}_i \in \mathcal{P}_\Omega} \sum_{t_j \in \mathcal{P}_t} \left(\frac{\partial u_{RN}(\boldsymbol{x}_i, t_j, \boldsymbol{p})}{\partial t} + \mathcal{L}(u_{RN}(\boldsymbol{x}_i, t_j, \boldsymbol{p})) - f(\boldsymbol{x}_i, t_j) \right)^2, \end{aligned} \tag{3.13}$$

que representa a soma dos quadrados dos resíduos nos pontos de $\mathcal{P}_\Omega \times \mathcal{P}_t$, em termos do vetor $\boldsymbol{p}$.

Quando J_R tende a zero, a solução u_{RN} tende a solução u. O treinamento da rede neural consiste exatamente no processo de minimização do funcional resíduo discreto (3.13), sujeito às restrições (3.11)-(3.12). Como J_R não depende do resultado desejado, que é a solução exata u, desconhecida a priori, o processo de treinamento da rede é não supervisionado. No treinamento não supervisionado o algoritmo backpropagation não pode ser utilizado porque o erro em cada unidade de saída não é disponível no processo de aprendizagem da rede (PARISI; MARIANI; LABORDE, 2003). Dessa forma, técnicas de otimização precisam ser utilizadas para minimizar J_R.

Obtido o vetor de parâmetros ótimos, $\boldsymbol{p}^*$, onde

$$\boldsymbol{p}^* = \arg \min_{\boldsymbol{p} \in \mathbb{R}^{n_p}} J_R(\boldsymbol{p}),$$

a solução aproximada $u_{RN}(\boldsymbol{x}, t, \boldsymbol{p}^*)$ é uma função suficientemente diferenciável no domínio $\Omega \times [0, T_f]$ (LAGARIS; LIKAS; FOTIADIS, 1998; PARISI; MARIANI; LABORDE, 2003).

3.0.1 Problema de Minimização

O problema de otimização que devemos resolver é da forma

$$\min_{\boldsymbol{p} \in \mathbb{R}^{n_p}} J_R(\boldsymbol{p}) \tag{3.14}$$

$$\text{sujeito à } \boldsymbol{F}(\boldsymbol{p}) = \boldsymbol{0}, \tag{3.15}$$

onde $J_R(\cdot)$ é dado por (3.13) e $\boldsymbol{F}$ é um operador que representa as restrições de igualdade (condições inicial e de contorno) dadas por (3.11)-(3.12). Nas seções 3.1 e 3.1 o problema de miminização restrita (3.14)-(3.15) será transformado em um problema de minimização irrestrita, que pode ser resolvido utilizando técnicas numéricas baseadas no gradiente. Métodos de otimização baseados no gradiente são procedimentos iterativos que utiliza informações do gradiente da função objetivo durante as iterações. Esses métodos podem ser descritos como segue: dados $k = 0$ e uma condição inicial $\boldsymbol{p}_0$, calcular

$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \alpha_k \boldsymbol{d}_k, \tag{3.16}$$

para $k = 1, 2, \cdots$ até um processo de parada for satisfeito. Em (3.16), α_k é o tamanho do passo na direção do vetor $\boldsymbol{d}_k = \boldsymbol{d}_k\Big(\boldsymbol{\nabla} J_R, \boldsymbol{p}_k\Big)$, que é uma função do gradiente $\boldsymbol{\nabla} J_R$ e da posição atual $\boldsymbol{p}_k$. Diferentes métodos utilizam diferentes formas de $\boldsymbol{d}_k\Big(\boldsymbol{\nabla} J_R, \boldsymbol{p}_k\Big)$.

Alguns dos mais importantes métodos de otimização, baseados no gradiente e usados no contexto de redes neurais artificias para resolver equações diferenciais, são: método do Gradiente ou da Descida Máxima (CURRY, 1944), método do Gradiente Estocástico (*Stochastic Descent Gradient*) (BOTTOU; CURTIS; NOCEDAL, 2018; SOYDANER, 2020; SUN et al., 2019), método Levenberg-Marquardt (HAGAN; MENHAJ, 1994), método Broyden-Fletcher-Goldfarb-Shanno (BFGS) (GUO; LIU; WANG, 2008), método Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) (LIU; NOCEDAL, 1989), ADAM (adaptive moment estimation) (KINGMA; BA, 2017). Vários desses métodos e suas extensões são implementados no TensorFlow. Uma revisão sobre métodos de otimização na área de aprendizagem de máquina pode ser obtida em (BOTTOU; CURTIS; NOCEDAL, 2018).

Um desafio particular em implementar esses algoritmos é determinar de forma eficiente as derivadas de J_R, que formam o vetor gradiente $\boldsymbol{\nabla}_{\boldsymbol{p}} J_R(\boldsymbol{p})$ com n_p componentes. Essas n_p derivadas podem ser obtidas numericamente pelo método de diferenças finitas. O TensorFlow e Torch determinam essas derivadas usando algoritmos de *Diferenciação Automática* (DA) (BAYDIN et al., 2018; MARGOSSIAN, 2019). DA é um conjunto de técnicas para calcular derivadas de funções em programas computacionais, que possuem várias vantagens em relação às técnicas clássicas (diferenças finitas) (BAYDIN et al., 2018).

3.1 Funcional resíduo com condições inicial e de contorno

A primeira abordagem no tratamento das condições inicial e de contorno consiste em incorporar essas restrições, através de uma função pênalti, no funcional resíduo (3.7) (DISSANAYAKE; PHAN-THIEN, 1994; RUDD, 2013). Esse esquema transforma o problema original de otimização restrita em um problema de otimização irrestrita. Segundo Rudd

(2013), como todos métodos que envolvem funções pênalties, essa estratégia pode exibir convergência lenta no processo iterativo e problemas de acurácia na satisfação das restrições de igualdade, que representam as condições inicial e de contorno. O problema da acurácia pode ser contornado através do uso de mais neurônios nas camadas escondidas e uma discretização mais refinada ao longo da fronteira do domínio, aumentando o custo computacional do procedimento numérico (RUDD, 2013). A seguir, apresentamos três exemplos dessa estratégia:

1. Dockhorn (2019) utilizam o funcional

$$J_R(\boldsymbol{p}) = \|R(u_{RN})\|^2_{L^2(\Omega\times[0,T_f])} + \lambda_1\|u_{RN} - g\|^2_{L^2(\Gamma\times[0,T_f])} + \lambda_2\|u_{RN}(\cdot,0,\cdot) - u_0\|^2_{L^2(\Omega)}, \tag{3.17}$$

   onde λ_1 e λ_2 são parâmetros de penalidade. Por exemplo, pode-se usar

$$\lambda_1 = 1/meas(\Gamma\times[0,T_f]) \quad \text{e} \quad \lambda_2 = 1/meas(\Omega).$$

   O primeiro termo do lado direito corresponde a minimização do resíduo da equação diferencial, o segundo termo é o tratamento das condições de contorno de Dirichlet e o último termo corresponde a condição inicial. Se $J_R(\boldsymbol{p}) \approx 0$ então $u_{RN} = u_{RN}(\boldsymbol{p})$ é uma solução aproximada do problema (3.1)-(3.3).

2. Sirignano e Spiliopoulos (2018) , Michoski et al. (2020) utilizam o funcional

$$J_R(\boldsymbol{p}) = \|R(u_{RN})\|^2_{L^2(\Omega\times[0,T_f])} + \|u_{RN} - g\|^2_{L^2(\Gamma\times[0,T_f])} + \|u_{RN}(\cdot,0,\cdot) - u_0\|^2_{L^2(\Omega)} \tag{3.18}$$

   na solução de um problema parabólico do tipo (3.1)-(3.3), que equivale a (3.17) com $\lambda_1 = \lambda_2 = 1$.

3. Bar e Sochen (2019) utilizam o funcional

$$J_R(\boldsymbol{p}) = \lambda\|R(u_{RN})\|^2_{L^2(\Omega)} + \mu\|R(u_{RN})\|_{L^\infty(\Omega)} + \|u_{RN} - g\|_{L^1(\Gamma)} + \theta(\boldsymbol{p}) \tag{3.19}$$

   na solução de um problema estacionário (não dependente do tempo). Os dois primeiros termos do lado direito de (3.19) são associados ao resíduo da equação diferencial. O primeiro termo minimiza o resíduo no sentido L^2, enquanto o segundo minimiza o resíduo máximo (norma do infinito). Esse termo é importante devido a norma L^2 avaliar o erro no sentido médio, não levando em conta os pontos de um conjunto de medida nula. A norma $\|R(u_{RN})\|_{L^\infty(\Omega)}$ é mais restritiva, capturando possíveis discrepâncias. O terceiro termo força a satisfação das condições de contorno de Dirichlet na fronteira Γ, enquanto o último termo, $\theta(\boldsymbol{p})$, é um termo regularizador dos parâmetros da rede neural, que pode ser ajustado, dependendo da aplicação. Os coeficientes positivos λ e μ ponderam a importância dos dois termos no processo de minimização de J_R.

Nesse tipo de abordagem, onde as condições inicial e de contorno são inseridas no funcional resíduo, a forma mais simples e mais utilizada para a solução aproximada u_{RN} é dada por

$$u_{RN} = u_{RN}(\boldsymbol{x}, t, N(\boldsymbol{x}, t, \boldsymbol{p})) = N(\boldsymbol{x}, t, \boldsymbol{p}), \tag{3.20}$$

ou seja, a solução aproximada é a própria função saída da rede neural.

Uma vantagem dessa estratégia é a facilidade de incorporar diferentes tipos de condições de contorno no funcional resíduo. Uma desvantagem é que as condições inicial e de contorno não são satifeitas de forma exata.

3.2 Funcional resíduo com a solução aproximada incorporando as condições inicial e de contorno

A outra abordagem, apresentada em (LAGARIS; LIKAS; FOTIADIS, 1998) e utilizada em (MCFALL; MAHAN, 2009; BEIDOKHTI; MALEK, 2009), consiste na construção de uma expressão para a solução aproximada, u_{RN}, que satisfaz as condições inicial e de contorno automaticamente, enquanto mantém uma rede neural que é treinada para minimizar um funcional quadrático representando o resíduo da equação diferencial. Segundo Rudd (2013), embora essa abordagem tenha se mostrado eficaz na solução de problemas de valor de contorno (PVC) com um alto grau de acurácia, ainda faltam estudos para uma melhor avaliação em problemas de valor de contorno e inicial (PVCI). Segundo McFall (2006), essa estratégia aprsenta melhores resultados do que àquela apresentada na Seção 3.1. Uma desvantagem dessa abordagem é que a expressão para a solução aproximada, envolvendo a satisfação automática das condições inicial e de contorno, depende do problema específico a ser resolvido e deve ser definida pelo usuário.

Nessa estratégia, a solução aproximada do problema (3.1)-(3.3) é da forma

$$u_{RN}(\boldsymbol{x}, t, N(\boldsymbol{x}, t, \boldsymbol{p})) = G(\boldsymbol{x}, t) + Q(\boldsymbol{x}, t, N(\boldsymbol{x}, t, \boldsymbol{p})), \tag{3.21}$$

onde a função $G(\boldsymbol{x}, t)$ é definida a priori de forma a satisfazer as condições inicial e de contorno, ou seja,

$$G(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \quad \forall (\boldsymbol{x}, t) \in \Omega \times (0, T_f], \tag{3.22}$$

$$G(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega. \tag{3.23}$$

A função G depende do problema a ser resolvido e deve ser definida pelo usuário. A função Q depende dos parâmetros da rede neural, descritos pelo vetor $\boldsymbol{p}$, e da função de ativação

utilizada, através da função N. Em geral, Q tem a forma

$$Q(\boldsymbol{x}, t, N(\boldsymbol{x}, t, \boldsymbol{p})) = \Phi(\boldsymbol{x}, t)N(\boldsymbol{x}, t, \boldsymbol{p}),$$

onde a função escalar $\Phi(\boldsymbol{x}, t)$ é escolhida de modo a não contribuir com as condições inicial e de contorno, isto é, $\Phi(\boldsymbol{x}, t) = 0$, para todo $(\boldsymbol{x}, t) \in \Gamma \times [0, T_f]$. Vale ressaltar que a função Q atua somente no interior do domínio Ω, para todo $t \in (0, T_f]$, não contribuindo com as condições inidial e de contorno, que são satisfeitas por $G(\cdot, \cdot)$. Pode existir mais de uma maneira de escolher as funções G e Φ. Uma vantagem dessa estratégia é que as condições de contorno são satisfeita de forma automática e exata. Vale ressaltar que as implementações numéricas dessa dissertação utilizam essa estratégia. Uma forma sistematica para determinar as funções G, Q e Φ para condições de contorno arbitrárias (Dirichlet, Neuman, Mista) sobre domínios retangulares são descrito em (LAGARIS; LIKAS; FOTIADIS, 1998). A Seção 3.4 apresenta alguns exemplos de escolhas das funções G, Q e Φ.

Essa estratégia não é simples se a fronteira tiver um formato mais complexo (não retangular), pois a descrição das funções Q e Φ, que não contribuem com as condições inicial e de contorno, podem se tornar muito difícil. Estratégias para usar essa abordagem em domínios com fronteiras irregulares são apresentadas em (LAGARIS; LIKAS; PAPAGEORGIOU, 2000; MCFALL; MAHAN, 2009).

3.3 A função saída da rede neural: $N(\mathbf{x}, t, \mathbf{p})$

Vale ressaltar que a rede neural utilizada nesse trabalho é do tipo *Perceptron* Multicamadas, apresentada no Capítulo 2.5. Muitos algoritmos de redes neurais para resolver equações diferenciais são descritos usando esse tipo de rede (LAGARIS; LIKAS; FOTIADIS, 1998; SILVA; NEITZEL; LIMA, 2008).

A solução aproximada u_{RN} é descrita em função de $N(\boldsymbol{x}, t, \boldsymbol{p})$. Inserindo as condições inicial e de contorno no funcional resíduo, resulta na escolha dada pela Eq. (3.20), isto é,

$$u_{RN}(\boldsymbol{x}, t, \boldsymbol{p}) = N(\boldsymbol{x}, t, \boldsymbol{p}).$$

No caso das condições de contorno serem satisfeitas pela própria solução u_{RN}, temos, de acordo com a Eq. (3.21),

$$u_{RN}(\boldsymbol{x}, t, N(\boldsymbol{x}, t, \boldsymbol{p})) = G(\boldsymbol{x}, t) + \Phi(\boldsymbol{x}, t)N(\boldsymbol{x}, t, \boldsymbol{p}).$$

Note-se que as principais propriedades de regularidade que a função u_{RN} possui advém da função $N(\cdot, \cdot, \cdot)$, que depende dos parâmetros de ajuste da rede neural. Como a

função u_{RN} satisfaz os operadores diferenciais $\frac{\partial(\cdot)}{\partial t}$ e $\mathcal{L}$, isso significa que a função N deve ser suficientemente diferenciável de forma que $\frac{\partial N}{\partial t}$ e $\mathcal{L}N$ existam. Mais do que isso, como a minimização do funcional resíduo é feita em termos do vetor de parâmetros $\boldsymbol{p}$, então N também deve ser suficientemente diferenciável em termos de $\boldsymbol{p}$, ou seja, em função dos parâmetros da rede (dados de entrada, pesos e limiares).

Para descrevermos a solução u_{RN} de forma explícita, precisamos determinar a função $N(\cdot,\cdot,\cdot)$. Para isso, considere rede neural da Fig. 8, composta por três camadas, camadas de *entrada*, *oculta* e de *saída*. Existem $d+1$ neurônios (nós) na camada de entrada, d_H neurônios (nós) na camada oculta, d_H limiares e um neurônio (nó) na saída. Dado o vetor de entrada $(\boldsymbol{x}, t) = (x_1, x_2, \cdots, x_d, t) \in \mathbb{R}^{d+1}$ da rede e o vetor de parâmetros de ajuste $\boldsymbol{p}$, a função N, para essa rede, é dada por

$$N(x_1, x_2, \cdots, x_d, t, \boldsymbol{p}) = \sum_{i=1}^{d_H} v_i \phi(z_i), \tag{3.24}$$

onde

$$z_i = \sum_{j=1}^{d} w_{ij} x_j + w_i t + l_i. \tag{3.25}$$

Além disso,

- $\boldsymbol{v} = (v_1, \cdots, v_{d_H})^T \in \mathbb{R}^{d_H}$ é o vetor dos pesos entre a camada oculta e a camada de saída, de forma que v_i corresponde ao peso entre o i-ésimo neurônio da camada oculta e o único neurônio da camada de saída;

- w_{ij} é um escalar que representa o peso entre o neurônio j da camada de entrada (j-ésimo componentes da entrada representando a componente da j-ésima direção espacial) com o i-ésimo neurônio da camada oculta; w_i é um escalar que representa o peso entre o dado t (tempo), de entrada, com o i-ésimo neurônio da camada oculta. Os pesos w_{ij} e w_i formam a matriz $W \in \mathbb{R}^{d_H \times (d+1)}$, conhecida como *matriz de pesos*;

- l_i é o valor limiar o i-ésimo neurônio da camada escondida;

- Os parâmetros de ajustes dados pelo vetor $\boldsymbol{v} = (v_1, \cdots, v_{d_H})^T \in \mathbb{R}^{d_H}$, pela matriz de pesos W e pelos limiares $l_1, \cdots, l_{d_H}$ formam o vetor $\boldsymbol{p}$. Observe que existem $d_H + d_H d + d_H + d_H = d_H(3+d)$ parâmetros ajustáveis, de forma que $\boldsymbol{p} \in \mathbb{R}^{d_H(d+3)}$ é dado por

$$\boldsymbol{p} = \boldsymbol{p}(w_{11}, \cdots, w_{1,d}, \cdots, w_{d_H,1}, \cdots, w_{d_H,d}, w_1, \cdots, w_{d_H}, v_1, \cdots, v_{d_H}, l_1, \cdots, l_{d_H}).$$

- $\phi : \mathbb{R} \longrightarrow \mathbb{R}$ é a função de ativação. No contexto de soluções numéricas de equações diferenciais, destacamos três possibilidades, dadas a seguir,

1. função de ativação sigmoidal:

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

2. função de ativação tangente hiperbólica:

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

3. função de ativação *Rectified Linear Unit* - RELU:

$$f(x) = max\{0, x\}.$$

O processo de minimização do funcional resíduo $J_R(\cdot)$, considerado o treinamento da rede neural, envolve o cálculo do seu gradiente (em função dos parâmetros de ajuste), e consequentemente, das derivadas parciais da função N em relação aos dados de entrada e dos parâmetros de ajuste. Nessa dissertação, essas derivadas são determinadas utilizando a técnica de diferenciação automática, implementada no TensorFlow. No Apêndice A apresentamos algumas derivadas da função N.

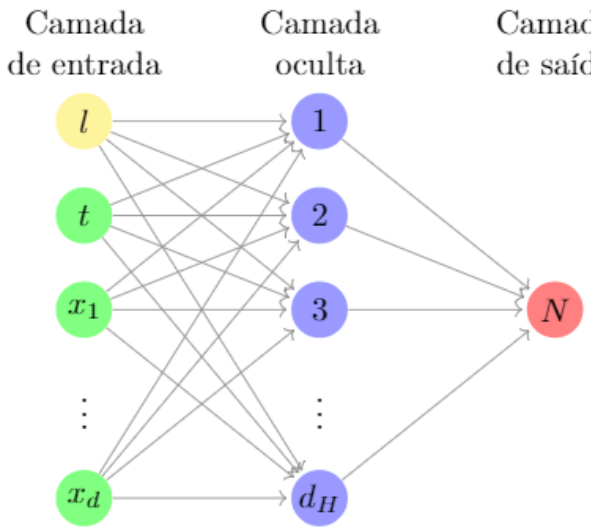


Figura 8 – Rede neural com três camadas: entrada (com $d+1$ neurônios), oculta (com d_H neurônios) e saída (com 1 neurônio). Os limiares (bias) da camada oculta são $l_1 = \cdots = l_{d_H} = l$.

3.4 Ilustração do método da Seção 3.2

Nessa seção vamos descrever através de alguns exemplos como definir a função de aproximação $u_{RN}(\boldsymbol{x}, t, \boldsymbol{p})$ de forma a satisfazer as condições inicial e de contorno. Além disso, mostraremos para cada caso, o funcional resíduo quadrático a ser minimizado.

3.4.0.1 Problema de valor inicial para equações diferenciais ordinárias

Vamos ilustrar a aplicação do método na solução de uma única equação diferencial ordinária. Considere o Problema de Valor Inicial (PVI) formado por uma EDO de primeira ordem,

$$\begin{aligned} \frac{du(t)}{dt} &= f(t, u(t)), \quad t \in (t_0, t_F]; && (3.26)\\ u(t_0) &= u_0, && (3.27) \end{aligned}$$

onde (3.27) representa a condição inicial. Para esse problema, a solução aproximada u_{RN} tem a forma (3.21) e é dada por

$$u_{RN}(t, \boldsymbol{p}) = u_0 + (t - t_0)N(t, \boldsymbol{p}), \tag{3.28}$$

isto é, $G(t) = u_0$ e $Q(t, N(t, \boldsymbol{p})) = \Phi(t)N(t, \boldsymbol{p})$, com $\Phi(t) = (t - t_0)$. Supondo N diferenciável, note que a função u_{RN}, descrita em (3.28), é uma função diferenciável (em t) e satisfaz automaticamente a condição inicial (3.27), por construção. Usando uma rede MCP com d_H neurônios na camada oculta, o vetor

$$\boldsymbol{p} = \boldsymbol{p}(w_1, \cdots, w_{d_H}, v_1, \cdots, v_{d_H}, l_1, \cdots, l_{d_H})$$

contém $3d_H$ parâmetros ajustáveis e usando a função de ativação sigmoidal,

$$\begin{aligned} N(t, \boldsymbol{p}) &= \sum_{k=1}^{d_H} v_k \phi(w_k t + l_k) \\ &= \sum_{k=1}^{d_H} \frac{v_k}{1 + e^{-(w_k t + l_k)}}, && (3.29) \end{aligned}$$

que é uma função diferenciável em termos de t, w_k, v_k e l_k. Então, a solução aproximada (3.28) resulta em

$$u_{RN}(t, \boldsymbol{p}) = u_0 + (t - t_0) \sum_{k=1}^{d_H} \frac{v_k}{1 + e^{-(w_k t + l_k)}}, \tag{3.30}$$

que também é uma função diferenciável em termos de t e dos parâmetros da rede.

O resíduo da equação diferencial é dado por

$$\begin{aligned} R(u_{RN}(t, \boldsymbol{p})) &= \frac{du_{RN}(t, \boldsymbol{p})}{dt} - f(t, u_{RN}(t, \boldsymbol{p})) \\ &= N(t, \boldsymbol{p}) + (t - t_0)\frac{\partial N(t, \boldsymbol{p})}{\partial t} - f(t, u_{RN}(t, \boldsymbol{p})) \end{aligned}$$

e o funcional resíduo discreto a ser minimizado é descrito por

$$J_R(\boldsymbol{p}) = \sum_{t_j \in \mathcal{P}_t} \left[N(t_j, \boldsymbol{p}) + (t_j - t_0)\frac{\partial N(t_j, \boldsymbol{p})}{\partial t} - f(t_j, u_{NR}(t_j, \boldsymbol{p})) \right]^2,$$

onde $\mathcal{P}_t$ é uma discretização do domínio temporal $[t_0, T_f]$.

Como a minimização do funcional J_R envolve o cálculo do gradiente de J_R,

$$\boldsymbol{\nabla} J_R(\boldsymbol{p}) = \left[\frac{\partial J_R}{\partial w_1} \quad \cdots \quad \frac{\partial J_R}{\partial w_{d_H}} \quad \frac{\partial J_R}{\partial v_1} \quad \cdots \quad \frac{\partial J_R}{\partial v_{d_H}} \quad \frac{\partial J_R}{\partial l_1} \quad \cdots \quad \frac{\partial J_R}{\partial l_{d_H}} \right]^T,$$

as seguintes derivadas, $\frac{\partial J_R}{\partial w_k}$, $\frac{\partial J_R}{\partial v_k}$ e $\frac{\partial J_R}{\partial l_k}$, $k = 1, \cdots, d_H$, devem ser determinadas. O TensorFlow determina essas derivadas utilizando derivação automática.

3.4.0.2 Problema de valor inicial para equações diferenciais ordinárias de segunda ordem

Considere agora o PVI formado por uma EDO de segunda ordem,

$$\begin{aligned} \frac{d^2u(t)}{dt^2} &= f\Big(t, u(t), \frac{du(t)}{dt}\Big), \quad t \in (t_0, T_f]; && (3.31)\\ u(t_0) &= u_0, && (3.32)\\ \frac{du(t_0)}{dt} &= u_0', && (3.33) \end{aligned}$$

onde (3.32) e (3.33) representam as condições iniciais. Para esse problema, a solução aproximada u_{RN} tem a forma (3.21) e é dada por

$$u_{RN}(t, \boldsymbol{p}) = u_0 + u_0'(t - t_0) + (t - t_0)^2 N(t, \boldsymbol{p}), \tag{3.34}$$

isto é, $G(t) = u_0 + u_0'(t - t_0)$ e $Q(t, N(t, \boldsymbol{p})) = \Phi(t)N(t, \boldsymbol{p})$, com $\Phi(t) = (t - t_0)^2$. Note que a função u_{RN}, descrita em (3.34), é uma função diferenciável (em t) e satisfaz automaticamente as condições iniciais (3.32)-(3.33), por construção. Usando uma rede MCP com d_H neurônios na camada oculta, o vetor

$$\boldsymbol{p} = \boldsymbol{p}(w_1, \cdots, w_{d_H}, v_1, \cdots, v_{d_H}, l_1, \cdots, l_{d_H})$$

contém $3d_H$ parâmetros ajustáveis e usando a função de ativação sigmoidal,

$$\begin{aligned} N(t, \boldsymbol{p}) &= \sum_{k=1}^{d_H} v_k \phi(w_k t + l_k)\\ &= \sum_{k=1}^{d_H} \frac{v_k}{1 + e^{-(w_k t + l_k)}}. \end{aligned} \tag{3.35}$$

Então a solução aproximada (3.34) resulta em

$$u_{RN}(t, \boldsymbol{p}) = u_0 + u_0'(t - t_0) + (t - t_0)^2 \sum_{k=1}^{d_H} \frac{v_k}{1 + e^{-(w_k t + l_k)}}. \tag{3.36}$$

Neste caso, o resíduo da equação é dado por

$$\begin{aligned} R(u_{RN}(t, \boldsymbol{p})) &= \frac{d^2 u_{RN}(t, \boldsymbol{p})}{dt^2} - f(t, u_{RN}(t, \boldsymbol{p}))\\ &= 2N(t, \boldsymbol{p}) + 4(t - t_0)\frac{\partial N(t, \boldsymbol{p})}{\partial t} + (t - t_0)^2 \frac{\partial^2 N(t, \boldsymbol{p})}{\partial t^2} - f(t, u_{RN}(t, \boldsymbol{p})) \end{aligned}$$

e o funcional resíduo discreto a ser minimizado é descrito por

$$J_R(\boldsymbol{p}) = \sum_{t_j \in \mathcal{P}_t} \left[2N(t_j, \boldsymbol{p}) + 4(t_j - t_0)\frac{\partial N(t_j, \boldsymbol{p})}{\partial t} + (t_j - t_0)^2 \frac{\partial^2 N(t_j, \boldsymbol{p})}{\partial t^2} - f(t_j, u_{RN}(t, \boldsymbol{p}))\right]^2, \tag{3.37}$$

onde $\mathcal{P}_i$ é uma discretização do domínio temporal $[t_0, T_f]$. O gradiente de J_R é descrito por

$$\boldsymbol{\nabla} J_R(\boldsymbol{p}) = \left[\frac{\partial J_R}{\partial w_1} \quad \cdots \quad \frac{\partial J_R}{\partial w_{d_H}} \quad \frac{\partial J_R}{\partial v_1} \quad \cdots \quad \frac{\partial J_R}{\partial v_{d_H}} \quad \frac{\partial J_R}{\partial l_1} \quad \cdots \quad \frac{\partial J_R}{\partial l_{d_H}} \quad \right]^T,$$

onde as derivadas, $\frac{\partial J_R}{\partial w_k}$, $\frac{\partial J_R}{\partial v_k}$ e $\frac{\partial J_R}{\partial l_k}$, $k = 1, \cdots, d_H$, são determinadas via diferenciação automática pelo TensorFlow.

3.4.0.3 PVC unidimensional

Se substituirmos as condições iniciais (3.32)-(3.33) por condições de contorno de Dirichlet,

$$u(a) = u_a, \tag{3.38}$$
$$u(b) = u_b, \tag{3.39}$$

o problema resultante é um PVC. Neste caso, a forma geral da solução aproximada (3.34) pode ser dada por

$$u_{RN}(x) = u_{RN}(x, \boldsymbol{p}) = u_a \frac{(b-x)}{(b-a)} + u_b \frac{(a-x)}{(a-b)} + (a-x)(b-x)N(x, \boldsymbol{p}), \tag{3.40}$$

isto é, $G(x) = u_a \frac{(b-x)}{(b-a)} + u_b \frac{(a-x)}{(a-b)}$ e $Q(x, N(x, \boldsymbol{p})) = (a-x)(b-x)N(x, \boldsymbol{p})$. Analogamente aos casos anteriores, determinamos o funcional resíduo discreto a ser minimizado.

3.4.0.4 Sistemas de equações diferenciais ordinárias de primeira ordem

Considere o sistema formado por m de EDOs de primeira ordem,

$$\begin{aligned} \frac{du_1}{dt} &= f_1(t, u_1, u_2, \cdots, u_m), \\ \frac{du_2}{dt} &= f_3(t, u_1, u_2, \cdots, u_m), \\ &\vdots \\ \frac{du_m}{dt} &= f_m(t, u_1, u_2, \cdots, u_m), \end{aligned}$$

com condições iniciais em $t = t_0$, dadas por

$$u_j(t_0) = C_j, \quad j = 1, 2, \cdots, m.$$

A forma geral da solução aproximada para esse sistema de EDOs é

$$u^j_{RN}(t, \boldsymbol{p}_j) = C_j + (t - t_0)N_j(t, \boldsymbol{p}_j), \quad j = 1, 2, \cdots, m. \tag{3.41}$$

Neste caso, é considerada uma rede neural para cada solução aproximada $u^j_{RN}(t, \boldsymbol{p}_j)$. O funcional resíduo discreto é dado por

$$J_R(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_m) = \sum_{j=1}^{m} \sum_{t_i \in \mathcal{P}_t} \left[\frac{du^j_{RN}(t_i, \boldsymbol{p}_j)}{dt} - f_j\Big(t_i, u^1_{RN}(t_i, \boldsymbol{p}_j), \cdots, u^m_{RN}(t_i, \boldsymbol{p}_m)\Big) \right]^2.$$

3.4.0.5 Equações Diferenciais Parciais

Considere o problema abstrato (bidimensional) que consiste em achar $u : \Omega \longrightarrow \mathbb{R}$ satisfazendo

$$\mathcal{L}u(x,y) = f(x,y), \quad (x,y) \in \Omega, \tag{3.42}$$

$$u(x,y) = g(x,y), \quad (x,y) \in \Gamma, \tag{3.43}$$

onde $\Omega \subset \mathbb{R}^2$ é um domínio limitado, retangular, com fronteira Γ, f é uma função conhecida, o termo de fonte, g é a condição de contorno de Dirichlet, também conhecida, e $\mathcal{L}$ é um operador diferencial (parcial) de segunda ordem.

Inserindo as condições de contorno na solução aproximada, conforme (3.21), temos

$$u_{RN}(x,y,\boldsymbol{p}) = G(x,y) + Q(x,y,N(\boldsymbol{x},\boldsymbol{p})),$$

onde a função $G(x,y)$ é definida a priori de forma a satisfazer as condições de contorno, ou seja,

$$G(x,y) = g(x,y), \quad \forall (x,y) \in \Gamma$$

e a função Q depende dos parâmetros da rede neural, descritos pelo vetor $\boldsymbol{p}$, e da função N. Neste caso, Q tem a forma

$$Q(x,y,N(x,y,\boldsymbol{p})) = \Phi(x,y)N(x,y,\boldsymbol{p}),$$

onde a função escalar Φ é escolhida de modo a não contribuir com as condições de contorno, isto é, $\Phi(x,y) = 0$, para todo $(x,y) \in \Gamma$. Considerando $\Omega = (L_a, L_b) \times (L_c, L_d)$, com $L_a < L_b$ e $L_c < L_c$, as funções G e Φ, associadas à solução aproximada do problema (3.42)-(3.43), podem ser definidas por

$$\begin{aligned} G(x,y) &= \frac{(L_b - x)}{(L_b - L_a)} g(L_a, y) + \frac{(x - L_a)}{(L_b - L_a)} g(L_b, y) \\ &+ \frac{(L_d - y)}{(L_b - L_a)(L_d - L_c)} \Big((L_b - L_a) g(x, L_c) - (L_b - x) g(L_a, L_c) - (x - L_a) g(L_b, L_c) \Big) \\ &+ \frac{(y - L_c)}{(L_b - L_a)(L_d - L_c)} \Big((L_b - L_a) g(x, L_d) - (L_b - x) g(L_a, L_d) - (x - L_a) g(L_b, L_d) \Big), \\ \Phi(x,y) &= (x - L_a)(L_b - x)(y - L_c)(L_d - y). \end{aligned}$$

Com essas escolhas de G e Φ, definimos a função aproximada u_{RN} e o funcional resíduo discreto é dado por

$$J_R(\boldsymbol{p}) = \sum_{(x_i,y_i) \in \mathcal{P}_\Omega} \Big[\mathcal{L}u_{RN}(x_i, y_i, \boldsymbol{p}) - f(x_i, y_i) \Big]^2. \tag{3.44}$$

Para o caso de condições de contorno de Dirichlet e Neumann, sugerimos a Referência (LAGARIS; LIKAS; FOTIADIS, 1998).

4 Experimentos Numéricos

Neste capítulo apresentamos um conjunto de experimentos numéricos envolvendo problemas estacionários e transientes, a fim de avaliar a qualidade das soluções obtidas pelo método baseado em redes neurais *perceptron* multicamadas. Algumas comparações, em termos de soluções gráficas e de erros medidos na norma do infinito, são realizadas com os métodos de elementos finitos e diferenças finitas. Utilizamos a estratégia apresentada na Seção 3.2, isto é, as condições inicial e de contorno são incorporadas na solução aproximada u_{RN}. Por simplicidade, a discretização dos domínios é feita como nos métodos numéricos clássicos (não utilizamos pontos aleatórios). Um algoritmo para resolver um PVIC bidimensional é descrito a seguir,

1. Inicialize o vetor de pesos (parâmetros) $\boldsymbol{p}$ e a taxa de aprendizagem η;
2. Gere um conjunto de pontos do domínio (discretização):
   - construa $\mathcal{P}_\Omega = \{(x_i, y_i) \in \Omega, i = 1, \cdots, n_\Omega\}$,
   - construa $\mathcal{P}_t = \{t_j \in [0, T], j = 1, \cdots, n_t\}$,
3. Descreva o funcional resíduo discreto $J_R(\boldsymbol{p})$;
4. Determine $\boldsymbol{p}^* \approx \min J_R(\boldsymbol{p})$ usando um método de minimização;
5. Obtém a solução u_{RN}, em função de $\boldsymbol{p}^*$;

Observação 4.0.1 *Utilizando um método baseado no gradiente no passo 4, o processo iterativo para determinar $\boldsymbol{p}^*$ é descrito por*

$$\boldsymbol{p}^{k+1} = \boldsymbol{p}^k - \eta \boldsymbol{\nabla}_{\boldsymbol{p}} J_R(\boldsymbol{p}),$$

onde η é a taxa de aprendizagem. Como critério de parada podemos utilizar,

$$\frac{\|\boldsymbol{p}^{k+1} - \boldsymbol{p}^{k+1}\|_\infty}{\|\boldsymbol{p}^{k+1}\|_\infty} \leq tol_1 \quad ou \quad \left|J_R(\boldsymbol{p})\right| \leq tol_2 \quad ou \quad k > k_{max}.$$

Nessa dissertação estamos utilizando o critério de parada $k > k_{max}$, onde na maioria dos experimentos, $k_{max} = 10000$.

As implementações computacionais foram realizadas utilizando um computador com processador Core i5, com 8 gigabytes de memória ram, em linguagem Python e com a

biblioteca TensorFlow do Google, o que proporciona algumas facilidades, tal como a utilização de funções de ativação e métodos de minimização previamente implementados, usando diferencição automática. O coeficiente de aprendizagem utilizado foi $\eta = 10^{-3}$.

4.1 TensorFlow para solução de equações diferenciais via redes neurais

O Tensorflow (ABADI et al., 2016) é um framework, de código aberto, desenvolvido em 2015 pelo Google, compatível com a linguagem Python e amplamente utilizado para o desenvolvimento de redes neurais profundas. Ele foi idealizado para ser flexível, eficiente, extensível e portátil. Além de ter um amplo suporte para aplicações na área de aprendizagem de máquina, também é adequado para realizar cálculos matemáticos complexos (ABRAHAMS et al., 2016).

O Tensorflow é descrito por um grafo, composto por um conjunto de nós, que configura um fluxo de dados. É necessário configurar os dados, as variáveis, os marcadores de posição (*placeholders*) e modelos (*models*) antes de informar ao algoritmo o início do treinamento. É criada uma função custo, o funcional resíduo, no caso de equações diferenciais, que o TensorFlow precisa minimizar, usando técnicas de otimização.

No Apêndice B apresentamos a implementação da solução do problema (4.1)-(4.2) usando o TensorFlow. Outras opções de bibliotecas de aprendizado de máquina com código aberto, são,

- *Theano* - é uma biblioteca de aprendizado de máquina extremamente flexível escrita em Python. Popularmente utilizado em pesquisa, com capacidade de definir modelos complexos com bastante facilidade;
- *Keras* - é uma biblioteca de alto nível baseada no TensorFlow, considerada mais simples de se trabalhar do que o próprio Tensorflow;
- *Torch* - é uma biblioteca de aprendizado de máquina com foco na implementação de GPU, escrito em linguagem Lua, apoiado por equipes de pesquisa em várias grandes empresas;
- *Chainer* - é uma biblioteca na linguagem Python para aprendizado de máquina, flexível e capaz de utilizar várias GPUs em uma única máquina.

4.2 Inicialização dos parâmetros

Os parâmetros ajustáveis de uma rede neural, pesos e limiares (bias), são otimizados durante a fase de treinamento, através da minimização do funcional resíduo (função custo). No entanto, eles devem ser inicializados antes de começar o processo de treinamento da rede, e essa etapa de inicialização tem um efeito importante no treinamento, em termos de sua convergência (KUMAR, 2017). O estudo de técnicas de inicialização dos parâmetros ajustáveis de uma rede neural é uma área de pesquisa ativa (ver (KUMAR, 2017) e as referências que ele cita). Uma das técnicas mais utilizada é conhecida como processo de *inicialização Xavier*, apresentado por Glorot e Bengio (2010). Vale destacar que o processo de *inicialização Xavier* é implementado no TensorFlow e será utilizado nessa dissertação.

4.3 Problemas Unidimensionais

4.3.1 EDO linear de primeira ordem

Nesse exemplo, consideramos uma EDO de primeira ordem associada ao PVI: achar a função $u : [x_0, x_f] \to \mathbb{R}$, satisfazendo

$$\begin{aligned} \frac{du(x)}{dx} + p(x)u(x) &= q(x), \quad x \in (x_0, x_f], && (4.1)\\ u(x_0) &= u_0, && (4.2) \end{aligned}$$

onde

$$x_0 = 0, x_f = 1, u_0 = 1, p(x) = x + \frac{1+3x^2}{1+x+x^3} \text{ e } q(x) = x^3 + 2x + x^2\Big(\frac{1+3x^2}{1+x+x^3}\Big).$$

Esse problema é estudado em (LAGARIS; LIKAS; FOTIADIS, 1998) e sua solução exata é

$$u(x) = \frac{-e^{\frac{x^2}{2}}}{(1+x+x^3)+x^2}.$$

A forma geral da solução aproximada para esse problema é

$$u_{RN}(x) = u_{RN}(x, \boldsymbol{p}) = u_0 + (x - x_0)N(x, \boldsymbol{p}) = 1 + xN(x, \boldsymbol{p})$$

e o funcional resíduo discreto a ser minimizado é dado por

$$J_R(\boldsymbol{p}) = \sum_{x_j \in \mathcal{P}_x} \Big[\frac{du_{RN}(x_j, \boldsymbol{p})}{dx} + p(x_j)u_{RN}(x_j, \boldsymbol{p}) - q(x_j)\Big]^2,$$

onde $\mathcal{P}_x$ é uma discretização do domínio temporal $[0, 1]$.

Os experimentos foram realizados utilizando o método de minimização Adam, com coeficiente de aprendizagem $\eta = 10^{-3}$, com função de ativação tangente hiperbólica e 10000 iterações. A Tabela 1 apresenta os erros, nas normas do infinito e $L^2(0,1)$, entre as soluções exata e aproximada obtida via redes neurais, assim como o valor mínimo do funcional resíduo discreto. Foi utilizado uma rede neural com uma única camada oculta, considerando seis quantidades diferentes de neurônios (5, 10, 30, 50, 100, 500) e dois conjuntos de pontos de treinamento (discretização do domínio), com $N_x = 10$ e $N_x = 20$ pontos, respectivamente. Em ambos casos, $N_x = 10$ e $N_x = 20$, o erro mínimo da ordem de 10^{-4} e o valor mínimo de J_R foram obtidos com 10 neurônios na camada oculta. A Tabela 2 apresenta os resultados das variáveis apresentadas na Tabela 1, porém utilizando uma rede neural com duas camadas ocultas (e mesmo número de neurônios por camada). Aumentando o número de camadas, os erros medidos não diminuem. Em particular, para $N_x = 10$, os erros mínimos (da ordem de 10^{-3}) foram obtidos com 30 neurônios em cada camada oculta. A Fig. 9 apresenta os gráficos das soluções exata e aproximada para uma rede com duas camadas ocultas, com 5 neurônios em cada uma delas e $N_x = 10$. Visualmente, as soluções coincidem, mas existe um erro da ordem de 10^{-3}, conforme mostra a Tabela 2.

	$N_x = 10$			$N_x = 20$		
Neurônios	$\|e\|_\infty$	$\|e\|_{L^2(0,1)}$	$\min J_R$	$\|e\|_\infty$	$\|e\|_{L^2(0,1)}$	$\min J_R$
5	$5,787 \times 10^{-4}$	$9,235 \times 10^{-4}$	$7,188 \times 10^{-8}$	$4,685 \times 10^{-4}$	$1,023 \times 10^{-3}$	$9,328 \times 10^{-9}$
10	$3,191 \times 10^{-4}$	$5,077 \times 10^{-4}$	$7,875 \times 10^{-10}$	$2,370 \times 10^{-4}$	$5,387 \times 10^{-4}$	$1,015 \times 10^{-9}$
30	$3,502 \times 10^{-3}$	$5,355 \times 10^{-3}$	$9,980 \times 10^{-6}$	$2,486 \times 10^{-3}$	$5,385 \times 10^{-3}$	$3,681 \times 10^{-7}$
50	$1,372 \times 10^{-3}$	$2,079 \times 10^{-3}$	$4,952 \times 10^{-7}$	$1,006 \times 10^{-3}$	$2,151 \times 10^{-3}$	$4,173 \times 10^{-7}$
100	$1,231 \times 10^{-3}$	$1,839 \times 10^{-3}$	$2,398 \times 10^{-7}$	$7,456 \times 10^{-4}$	$1,518 \times 10^{-3}$	$2,369 \times 10^{-8}$
500	$1,208 \times 10^{-3}$	$1,821 \times 10^{-3}$	$1,719 \times 10^{-7}$	$9,276 \times 10^{-4}$	$2,008 \times 10^{-3}$	$4,899 \times 10^{-9}$

Tabela 1 – Erros ($e = u - u_{RN}$) nas normas do infinito e $L^2(0,1)$; valor mínimo do funcional J_R - considerando 6 quantidades diferentes de neurônios em duas camadas ocultas e duas discretizações, com $N_x = 10$ e $N_x = 20$.

	$N_x = 10$			$N_x = 20$		
Neurônios	$\|e\|_\infty$	$\|e\|_{L^2(0,1)}$	$\min J_R$	$\|e\|_\infty$	$\|e\|_{L^2(0,1)}$	$\min J_R$
5	$3,773 \times 10^{-3}$	$5,951 \times 10^{-3}$	$5,366 \times 10^{-7}$	$2,851 \times 10^{-3}$	$6,361 \times 10^{-3}$	$1,878 \times 10^{-6}$
10	$2,320 \times 10^{-3}$	$3,880 \times 10^{-3}$	$1,052 \times 10^{-6}$	$1,673 \times 10^{-3}$	$3,785 \times 10^{-3}$	$3,645 \times 10^{-7}$
30	$1,673 \times 10^{-3}$	$3,785 \times 10^{-3}$	$3,645 \times 10^{-7}$	$2,676 \times 10^{-3}$	$5,820 \times 10^{-3}$	$6,203 \times 10^{-8}$
50	$4,025 \times 10^{-3}$	$6,161 \times 10^{-3}$	$2,300 \times 10^{-5}$	$2,703 \times 10^{-3}$	$5,861 \times 10^{-3}$	$3,875 \times 10^{-7}$
100	$4,461 \times 10^{-3}$	$6,954 \times 10^{-3}$	$9,293 \times 10^{-8}$	$3,026 \times 10^{-3}$	$6,570 \times 10^{-3}$	$2,687 \times 10^{-6}$
500	$4,356 \times 10^{-3}$	$6,767 \times 10^{-3}$	$4,939 \times 10^{-8}$	$3,002 \times 10^{-3}$	$6,515 \times 10^{-3}$	$2,755 \times 10^{-6}$

Tabela 2 – Erros ($e = u - u_{RN}$) nas normas do infinito e $L^2(0,1)$; valor mínimo do funcional J_R - considerando 6 quantidades diferentes de neurônios em uma camada oculta e duas discretizações, com $N_x = 10$ e $N_x = 20$.

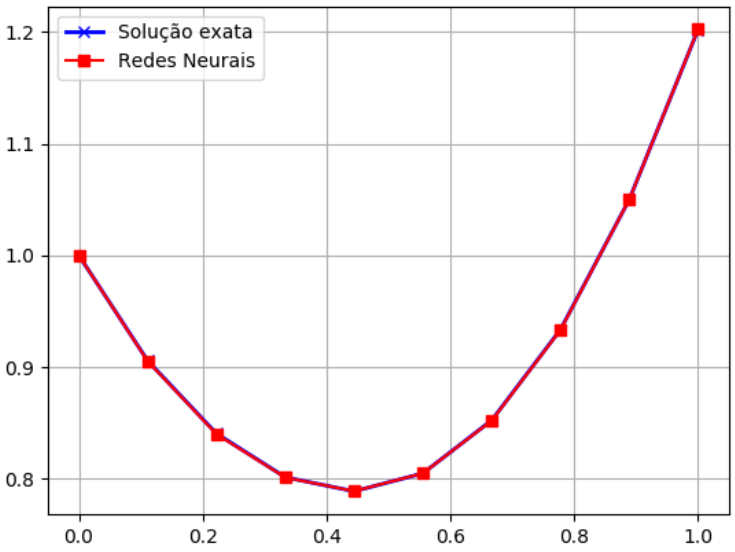


Figura 9 – Soluções exata e aproximada, para uma rede com 2 camadas ocultas com 5 neurônios cada uma delas e $N_x = 10$.

Com o objetivo de comparar as soluções obtidas por redes neurais, métodos de diferenças finitas Euler explícito e Runge Kutta de 2ª ordem, a Tabela 3 apresenta os erros na norma do infinito para essas três metodologias, usando discretizações com $N_x = 10, 20, 40$. Foi utilizada uma rede neural com duas camadas ocultas e 50 neurônios cada uma delas. A qualidade da solução via redes neurais é comparável com àquela obtida pelo método Runge-Kutta de 2ª ordem, conforme também a Fig. 10.

Métodos	$\|e\|_\infty$		
	$N_x = 10$	$N_x = 20$	$N_x = 40$
Euler explícito	$1,359 \times 10^{-1}$	$6,214 \times 10^{-2}$	$2,979 \times 10^{-2}$
Runge-Kutta 2ª ordem	$3,614 \times 10^{-3}$	$7,698 \times 10^{-4}$	$1,783 \times 10^{-4}$
Redes neurais	$1,372 \times 10^{-3}$	$1,006 \times 10^{-4}$	$7,783 \times 10^{-4}$

Tabela 3 – Erros, na norma do infinito, dos métodos Euler explícito, Runge-Kutta de 2ª ordem e redes neurais (com duas camadas ocultas e 50 neurônios cada camada), considerando três discretizações: $N_x = 10$, $N_x = 20$ e $N_x = 40$.

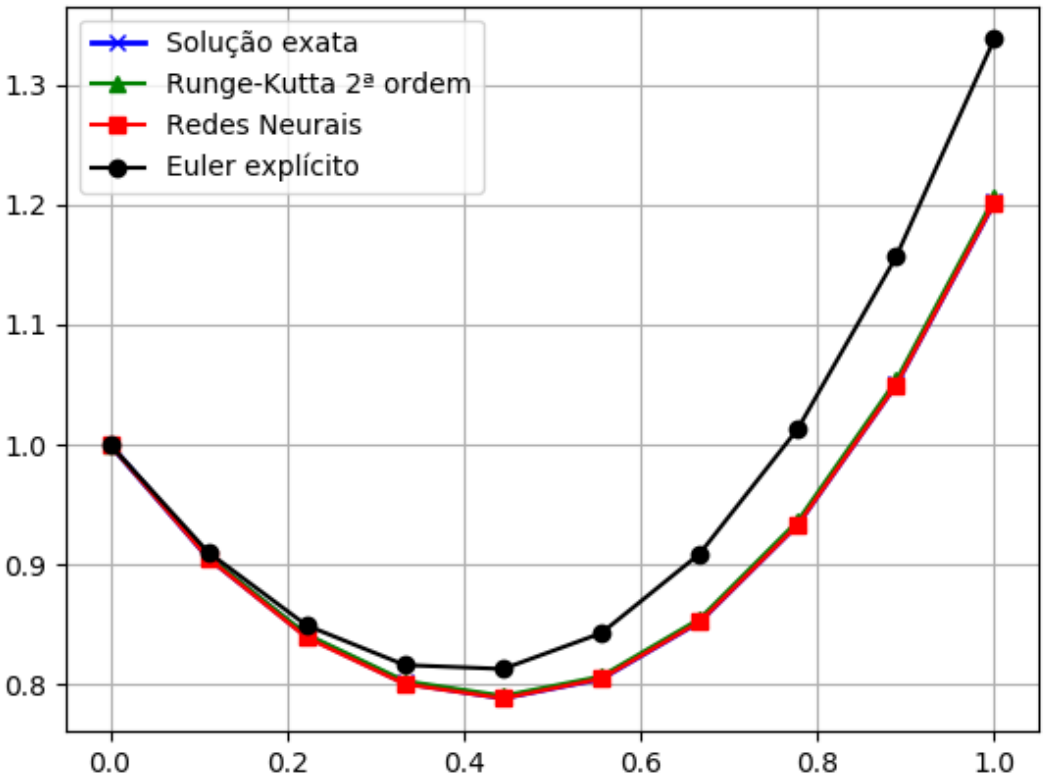


Figura 10 – Soluções obtidas com os métodos: Euler explícito, Runge Kutta de 2ª ordem e Redes neurais (com uma camada oculta com 50 neurônios), utilizando uma discretização com $N_x = 10$.

4.3.2 EDO linear de segunda ordem

Nesse exemplo, vamos considerar uma EDO de segunda ordem associada ao PVI, que consiste em encontrar a função $u : [x_0, x_f] \to \mathbb{R}$ satisfazendo

$$\begin{aligned} \frac{d^2u(x)}{dx^2} + \alpha\frac{du(x)}{dx} + u(x) &= q(x), \quad x \in (x_0, x_f], && (4.3)\\ u(x_0) &= u_0, && (4.4)\\ \frac{du(x_0)}{dx} &= u_1, && (4.5) \end{aligned}$$

com $x_0 = 0$, $x_f = 2$, $u_0 = 0$, $u_1 = 1$, $\alpha = \frac{1}{5}$ e $q(x) = -\frac{1}{5}e^{-x/5}cos(x)$ e cuja solução exata é

$$u(x) = e^{-x/5}sen(x). \tag{4.6}$$

A forma geral da solução aproximada para esse problema, descrita em (3.34), é dada por

$$u_{RN}(x) = u_{RN}(x, \boldsymbol{p}) = u_0 + u_1(x - x_0) + (x - x_0)^2 N(x, \boldsymbol{p}) = x + x^2 N(x, \boldsymbol{p})$$

e o funcional resíduo discreto a ser minimizado é dado por

$$J_R(\boldsymbol{p}) = \sum_{x_j \in \mathcal{P}_x} \left[\frac{du_{RN}^2(x_j, \boldsymbol{p})}{dx^2} + \alpha \frac{du_{RN}(x_j, \boldsymbol{p})}{dx} + u_{RN}(x_j, \boldsymbol{p}) - q(x_j) \right]^2,$$

onde $\mathcal{P}_x$ é uma discretização do domínio temporal $[0, 2]$.

Para a solução deste problema foram utilizados os seguintes parâmetros, Descida do gradiente como método de minimização, coeficiente de aprendizagem $\eta = 10^{-3}$, função de ativação sigmóide e 10000 iterações. A Tabela 1 apresenta os erros, na norma do infinito entre as soluções exata e aproximada obtida via redes neurais, bem como o valor mínimo do funcional resíduo discreto. Foi utilizado uma rede neural com uma única camada oculta, considerando seis quantidades diferentes de neurônios (5, 10, 30, 50, 100 e 500) e três conjuntos de pontos de treinamento, com $N_x = 10$, $N_x = 20$ e $N_x = 40$ pontos, respectivamente.

Para todos os casos, $N_x = 10$,$N_x = 20$ e $N_x = 40$, o erro mínimo é da ordem de 10^{-4} e o valor mínimo de J_R foram obtidos com 10 neurônios na camada oculta, onde o valor mínimo de J_R chega a 10^{-9} para $N_x = 40$.Observou-se que utilizando apenas uma camada oculta, o aumento no número de neurônios é inversamente proporcional ao erro, independentemente do número de pontos da discretização.

A Fig. 11 apresenta o gráfico das soluções exata e aproximada para uma rede de 1 camada oculta com 10 neurônios e $N_x = 10$. À primeira vista, as soluções coincidem,no entanto, existe um erro da ordem de 10^{-4}, conforme mostra a Tabela 4.

Neurônios	$N_x = 10$		$N_x = 20$		$N_x = 40$	
	$\|e\|_\infty$	$\min J_R$	$\|e\|_\infty$	$\min J_R$	$\|e\|_\infty$	$\min J_R$
5	$9,445 \times 10^{-4}$	$8,250 \times 10^{-6}$	$7,768 \times 10^{-4}$	$8,742 \times 10^{-7}$	$3,669 \times 10^{-2}$	$1,872 \times 10^{-7}$
10	$9,385 \times 10^{-4}$	$8,322 \times 10^{-6}$	$1,446 \times 10^{-3}$	$2,670 \times 10^{-7}$	$4,163 \times 10^{-2}$	$6,603 \times 10^{-9}$
30	$6,520 \times 10^{-4}$	$9,240 \times 10^{-7}$	$6,104 \times 10^{-2}$	$1,618 \times 10^{-4}$	$2,502 \times 10^{-2}$	$1,756 \times 10^{-7}$
50	$9,191 \times 10^{-3}$	$7,921 \times 10^{-5}$	$3,806 \times 10^{-2}$	$2,134 \times 10^{-6}$	$3,023 \times 10^{-2}$	$2,563 \times 10^{-5}$
100	$1,205 \times 10^{-1}$	$1,759 \times 10^{-4}$	$2,971 \times 10^{-2}$	$6,912 \times 10^{-5}$	$3,357 \times 10^{-2}$	$4,924 \times 10^{-5}$
500	$4,943 \times 10^{-2}$	$6,968 \times 10^{-4}$	$2,998 \times 10^{-2}$	$6,706 \times 10^{-5}$	$5,056 \times 10^{-2}$	$7,107 \times 10^{-6}$

Tabela 4 – Erros ($e = u - u_{RN}$) na norma do Infinito; valor mínimo do funcional J_R - considerando 6 quantidades diferentes de neurônios em uma camada oculta e duas discretizações, com $N_x = 10$, $N_x = 20$ e $N_x = 40$.

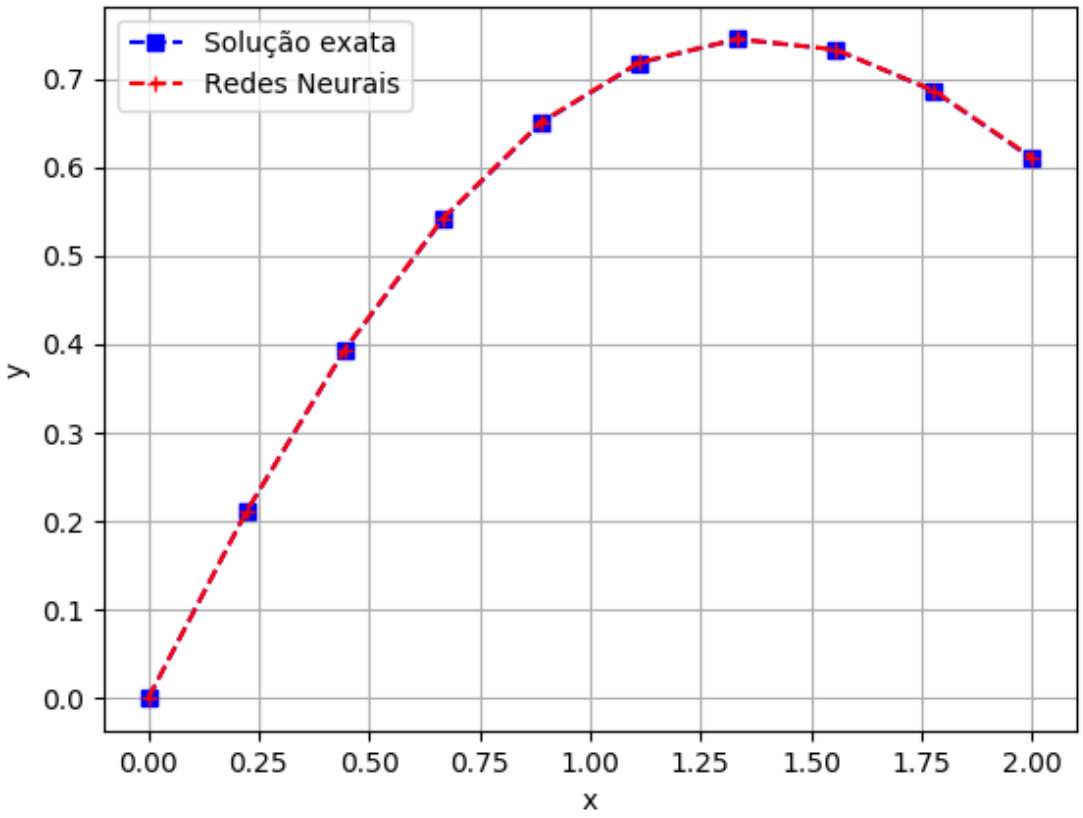


Figura 11 – Comparação entre a solução exata e a solução obtida por Redes neurais (com uma camada oculta com 10 neurônios), utilizando uma discretização com $N_x = 10$.

4.4 Problema Bidimensional

4.4.1 Problema de Poisson

Nesse sessão aplicaremos a metodologia baseada em redes neurais para resolver um problema bidimensional em domínios retangulares. Considere o problema de Poisson, que pode ser descrito por (3.42)-(3.43), com

$$\mathcal{L}u(x,y) = -\Delta u(x,y) = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$

isto é,

$$-\Delta u(x,y) = f(x,y), \quad (x,y) \in \Omega, \tag{4.7}$$

$$u(x,y) = g(x,y), \quad (x,y) \in \Gamma. \tag{4.8}$$

Esse problema bidimensional, do ponto de vista matemático, é do tipo elíptico. Neste caso, os métodos numéricos clássicos, como elementos finitos de Galerkin e diferenças finitas, se comportam bem quando aplicados em sua solução (QUARTERONI, 2009). Nosso objetivo aqui é avaliar o comportamento do método baseado em redes neurais PMC na solução desse problema. Vamos considerar um problema de Poisson, descrito no domínio quadrado $\Omega = (0,1) \times (0,1)$, com solução exata

$$u(x,y) = \operatorname{sen}(\pi x)\operatorname{sen}(\pi y).$$

As funções f e g são obtidas de maneira que a solução exata satisfaça o problema (4.7)-(4.8), isso é, $f(x,y) = 2\pi^2 \operatorname{sen}(\pi x)\operatorname{sen}(\pi y)$ em Ω e $g(x,y) = \operatorname{sen}(\pi x)\operatorname{sen}(\pi y)$ em Γ.

	Neurônios		
Camadas ocultas	10	20	40
1	$2,601 \times 10^{-2}$	$8,075 \times 10^{-2}$	$1,563 \times 10^{-2}$
2	$2,632 \times 10^{-2}$	$8,254 \times 10^{-2}$	$5,082 \times 10^{-3}$
3	$2,592 \times 10^{-2}$	$8,131 \times 10^{-2}$	$9,738 \times 10^{-3}$
4	$2,632 \times 10^{-2}$	$8,258 \times 10^{-2}$	$5,093 \times 10^{-3}$
5	$2,510 \times 10^{-2}$	$7,979 \times 10^{-2}$	$6,382 \times 10^{-3}$

Tabela 5 – Erros ($e = u - u_{RN}$) na norma do infinito, considerando 3 quantidades diferentes de neurônios e até 5 camadas ocultas. Foi utilizada uma discretização com $N_x = N_y = 10$, isto é, 100 pontos.

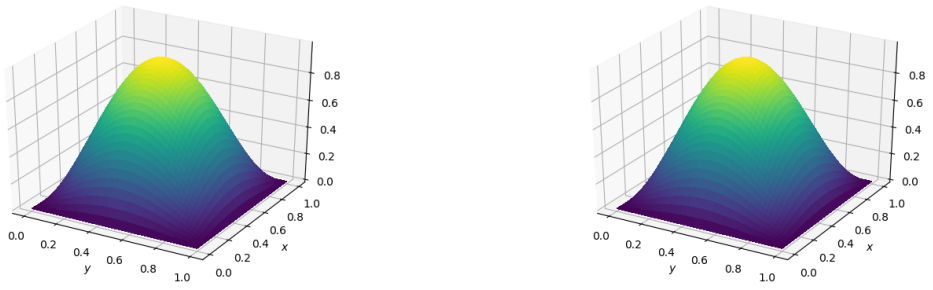
	Neurônios		
Camadas ocultas	20	50	100
1	$4,692 \times 10^{-3}$	$8,439 \times 10^{-3}$	$1,061 \times 10^{-2}$
2	$1,575 \times 10^{-3}$	$1,251 \times 10^{-3}$	$2,380 \times 10^{-3}$
3	$1,985 \times 10^{-3}$	$8,183 \times 10^{-4}$	$6,724 \times 10^{-4}$
4	$1,660 \times 10^{-3}$	$1,038 \times 10^{-3}$	$6,185 \times 10^{-4}$
5	$2,326 \times 10^{-3}$	$1,362 \times 10^{-3}$	$9,118 \times 10^{-4}$

Tabela 6 – Erros ($e = u - u_{RN}$) na norma do infinito, considerando 3 quantidades diferentes de neurônios e até 5 camadas ocultas. Foi utilizada uma discretização com $N_x = N_y = 50$, isto é, 2500 pontos.

Resolvemos esse problema usando redes neurais, método de minimização foi a Descida do gradiente, coeficiente de aprendizagem $\eta = 10^{-3}$, função de ativação tangente hiperbólica e 10000 iterações, posteriormente comparamos os resultados obtidos com a solução exata. A Tabela 5 mostra os erros na norma do infinito considerando até 5 camadas ocultas e 3 quantidades diferentes de neurônios por camada (10, 20 e 40). A rede foi treinada em um conjunto relativamente pequeno de pontos, $10 \times 10 = 100$ pontos. Observamos que usando 10,

20 ou 40 neurônios, o erro não decresce quando aumentamos o número de camadas ocultas. O menor erro foi obtido com duas camadas ocultas e 40 neurônios em cada uma delas. A Fig. 12 apresenta os gráficos das soluções exata e aproximada, com 1 camada e 10 neurônios.

Escolhemos um conjunto maior de pontos de treinamento para verificar o comportamento do erro na norma do infinito. A Tabela apresenta os erros considerando um conjunto de treinamento com $50 \times 50 = 2500$ pontos. Além disso, escolhemos 20, 50 e 100 neurônios em cada camada oculta.



(a) Solução exata (b) Solução RNA

Figura 12 – Soluções exata e RNA para o problema de Poisson, com uma camada oculta com 10 neurônios, utilizando uma discretização com $N_x = N_y = 50$.

4.5 Problemas transientes

4.5.1 Equação parabólica

A equação diferencial parcial parabólica que estudaremos é a equação do calor ou equação de difusão (transiente),

$$\frac{\partial u}{\partial t}(x,t) - \alpha \frac{\partial^2 u}{\partial x^2}(x,t) = f(x,t), \quad x \in (0,L),\ t > 0, \tag{4.9}$$

sujeita às condições de contorno (de Dirichlet),

$$u(0,t) = u_0^0(t), \quad u(L,t) = u_L^0(t), \quad \forall t > 0 \tag{4.10}$$

e condição inicial

$$u(x,0) = u_0(x), \quad \forall x \in [0,L]. \tag{4.11}$$

Em (4.9), f é uma função dada (termo de fonte) e $\alpha > 0$ é uma constante chamada de coeficiente de difusividade térmica, para problemas que envolvem distribuição de temperatura,

ou coeficienete de difusão, para problemas que envolvem a concentração de substâncias (difusão de espécies).

Em problemas que envolvem distribuição de temperatura, a Eq. (4.9) modela a distribuição de temperatura na região unidimensional (uma viga por exemplo) $[0, L]$, com condições de contorno dadas por (4.10) (temperaturas conhecidas na extremidade da viga) e condição inicial descrita em (4.11). A função $u(x, t)$ representa a temperatura no ponto $x \in [0, L]$ no tempo $t \geq 0$.

Considerando $f = 0$, condições de contorno de Dirichlet homogêneas, isto é, $u_0^0(t) = u_L^0(t) = 0$, para todo $t > 0$ e condição inicial

$$u(x, 0) = u_0(x) = sen\left(\frac{\pi x}{L}\right), \quad \forall x \in [0, L], \tag{4.12}$$

a solução exata para o problema (4.9), (4.10) e (4.11) é dada por

$$u(x, t) = e^{\frac{-\alpha\pi^2 t}{L}} sen\left(\frac{\pi x}{L}\right). \tag{4.13}$$

Em geral, os métodos numéricos clássicos para resolver problemas que evoluem no tempo utilizam duas discretizações, uma no espaço e outra no tempo. Aqui, utilizaremos uma única abordagem de solução, considerando como domínio de treinamento, o domínio espaço-tempo,

$$\Omega = (0, L) \times (0, T],$$

onde $T > 0$ é o tempo final. Dessa forma, a condição inicial é tratada como uma condição de contorno.

A solução aproximada é dada por

$$u_{RN}(x, t, \boldsymbol{p}) = G(x, t) + Q(x, t, N(\boldsymbol{x}, \boldsymbol{p})),$$

onde a função $G(x, t)$ é definida a priori de forma a satisfazer as condições de contorno e inicial, ou seja,

$$G(x, t) = \begin{cases} 0, & \text{se } x = 0 \text{ ou } x = L; \\ u_0(x), & \text{se } t = 0, \end{cases}$$

onde $u_0(x) = sen\left(\frac{\pi x}{L}\right)$ é a condição inicial e a função Q depende dos parâmetros da rede neural, descritos pelo vetor $\boldsymbol{p}$, e da função N. Neste caso, Q tem a forma

$$Q(x, t, N(x, t, \boldsymbol{p})) = \Phi(x, t) N(x, t, \boldsymbol{p}),$$

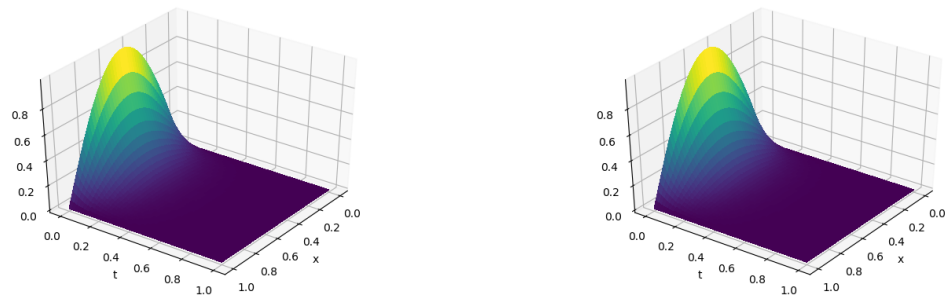
onde a função escalar Φ é escolhida de modo a não contribuir com as condições de contorno e inicial. As funções G e Φ são definidas por

$$\begin{aligned} G(x, t) &= \frac{(L - t)}{L} u_0(x), \\ \Phi(x, t) &= x(L - x)t. \end{aligned}$$

Note que $G(x,0) = u_0(x)$, $G(0,t) = \frac{(L-t)}{L}u_0(0) = 0$ e $G(L,t) = \frac{(L-t)}{L}u_0(L) = 0$. Analogamente, $\Phi(0,t) = \Phi(L,t) = \Phi(x,0) = 0$. Com essas escolhas, definimos a função aproximada u_{RN} e construímos o funcional resíduo discreto,

$$J_R(\boldsymbol{p}) = \sum_{(x_i,t_i)\in\mathcal{P}_\Omega} \left[\frac{\partial u_{RN}(x_i,t_i,\boldsymbol{p})}{\partial t} - \alpha\frac{\partial^2 u_{RN}(x_i,t_i,\boldsymbol{p})}{\partial x^2} - f(x_i,t_i)\right]^2. \tag{4.14}$$

Este problema foi resolvido usando um processo de aprendizagem com 10000 iterações, taxa de aprendizagem $\eta = 0,001$, função de ativação sigmóide e o método de minimização *Adam*. Foram comparados os resultados obtidos com a solução exata (4.13) e solução obtida através da rede neural (ver Fig. 13), enquanto que a Fig. 14 faz comparações entre as duas soluções para diferentes medidas de tempo. A Tabela 7 mostra os erros nas normas do Infinito para várias configurações da rede neurais (camadas ocultas e neurônios por camada). Nesse experimento estamos considerando $L = \alpha = 1$. Usando 10, 20 e 40 neurônios em até 5 camadas, o erro variou na ordem de 10^{-1} a 10^{-4}. Quando se utiliza pelo menos 4 camadas com 20 neurônios ou mais, o erro decresce na ordem de 10^{-4}. O menor erro foi obtido usando 5 camadas ocultas com 40 neurônios em cada uma delas.



(a) Solução exata da equação (b) Solução por redes neurais artificiais

Figura 13 – Comparação da solução exata com a solução via Redes Neurais Artificiais de (4.9), com duas camadas ocultas com 20 neurônios, utilizando uma discretização com $N_x = N_y = 50$.

.

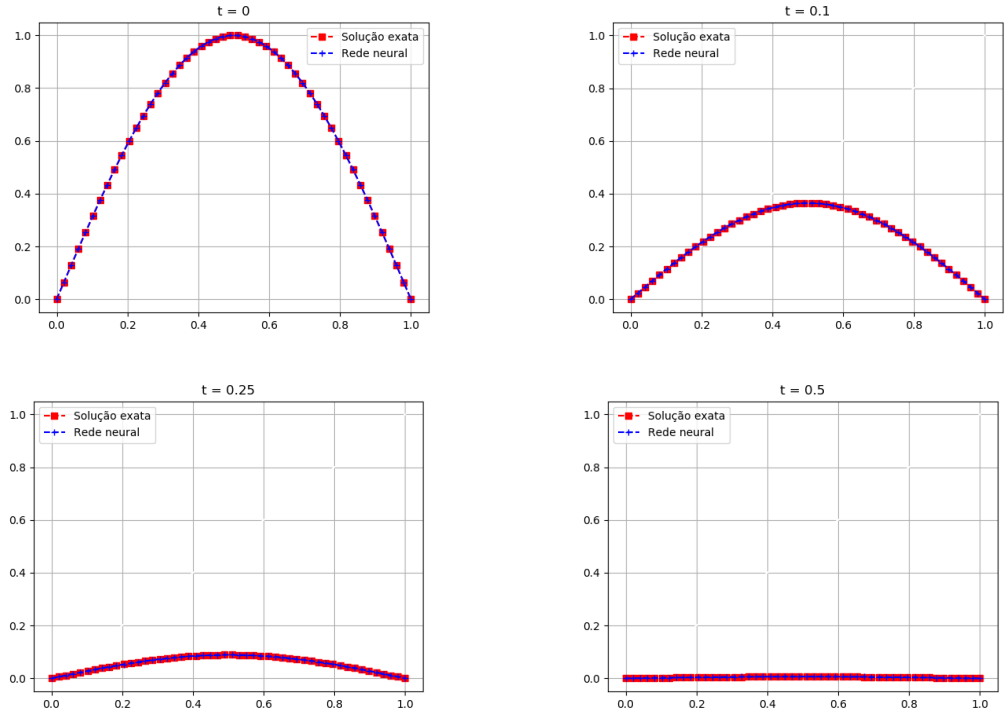


Figura 14 – Soluções exatas e por Redes neurais de 4.9 em diferentes medidas de tempo, com duas camadas ocultas com 40 neurônios, utilizando uma discretização com $N_x = N_y = 40$.

Camadas	Neurônios		
	10	20	40
1	1.073×10^{-1}	3.551×10^{-2}	4.164×10^{-2}
2	1.374×10^{-1}	1.910×10^{-3}	1.363×10^{-3}
3	1.693×10^{-2}	1.318×10^{-1}	1.193×10^{-3}
4	1.375×10^{-1}	$\underline{5.878 \times 10^{-4}}$	2.351×10^{-1}
5	$\underline{1.525 \times 10^{-3}}$	1.118×10^{-3}	$\underline{9.380 \times 10^{-4}}$

Tabela 7 – Erros na norma do infinito considerando quantidades diferentes de neurônios em até 5 camadas ocultas e discretização com $N_x = 50$ e $N_t = 50$.

4.5.2 Equação hiperbólica

Considere o problema hiperbólico, descrito pela equação das ondas em uma dimensão,

$$\frac{\partial^2 u(x,t)}{\partial t^2} - c^2 \frac{\partial^2 u(x,t)}{\partial x^2} = 0, \quad x \in (0, L), \ t > 0, \tag{4.15}$$

juntamente com condições iniciais e de contorno apropriadas, onde c, a velocidade da onda, é constante. Essa equação descreve as (pequenas) oscilações longitudinais de uma barra elástica

e homogênea (ou uma corda) com comprimento $L > 0$, sem a influência de forças externas. Considerando as extremidades da barra elástica (ou da corda) fixa, podemos descrever condições de contorno de Diriclet,

$$u(0,t) = u(1,t) = 0, \quad t \geq 0. \tag{4.16}$$

As condições iniciais são descritas pelo deslocamento inicial da corda, representado por $u(x,0)$, e pela velocidade inicial, $\frac{\partial u(x,t)}{\partial t}\Big|_{t=0}$, isto é,

$$u(x,0) = w(x), \quad x \in [0,L], \tag{4.17}$$

$$\frac{\partial u(x,t)}{\partial t}\Big|_{t=0} = v(x), \quad x \in [0,L]. \tag{4.18}$$

Considere o problema com solução exata,

$$u(x,t) = \operatorname{sen}(\pi x)\Big[\cos(\pi t) - \operatorname{sen}(\pi t)\Big], \tag{4.19}$$

de forma que $c = 1$, $w(x) = \operatorname{sen}(\pi x)$ e $v(x) = -\pi \operatorname{sen}(\pi x)$.

A solução aproximada, via redes neurais, é da forma

$$u_{RN}(x,t,\boldsymbol{p}) = G(x,t) + \Phi(x,t)N(x,t,\boldsymbol{p}),$$

onde

$$\begin{aligned} G(x,t) &= (1-t^2)w(x) + tv(x), \\ \Phi(x,t) &= x(1-x)t^2. \end{aligned}$$

Observe que com essa escolha das funções G e Φ, a solução aproximada, u_{RN}, satisfaz as condições iniciais e de contorno.

	Neurônios		
Camadas	10	20	40
1	$\underline{3.211 \times 10^{-3}}$	$\underline{1.996 \times 10^{-3}}$	1.321×10^{-3}
2	3.464×10^{-2}	7.817×10^{-2}	7.826×10^{-2}
3	3.382×10^{-3}	6.284×10^{-2}	7.151×10^{-2}
4	5.025×10^{-2}	7.825×10^{-2}	1.323×10^{-3}
5	7.825×10^{-2}	7.000×10^{-2}	$\underline{3.432 \times 10^{-4}}$

Tabela 8 – Erros na norma do infinito considerando quantidades diferentes de neurônios em até 5 camadas ocultas e discretização com $N_x = 50$ e $N_t = 50$.

(a) Solução exata da equação.

(b) Solução por redes neurais artificiais.

Figura 15 – Comparação da solução exata com a solução via Redes Neurais Artificiais de (4.15),com uma camada oculta com 10 neurônios, utilizando uma discretização com $N_x = N_y = 40$.

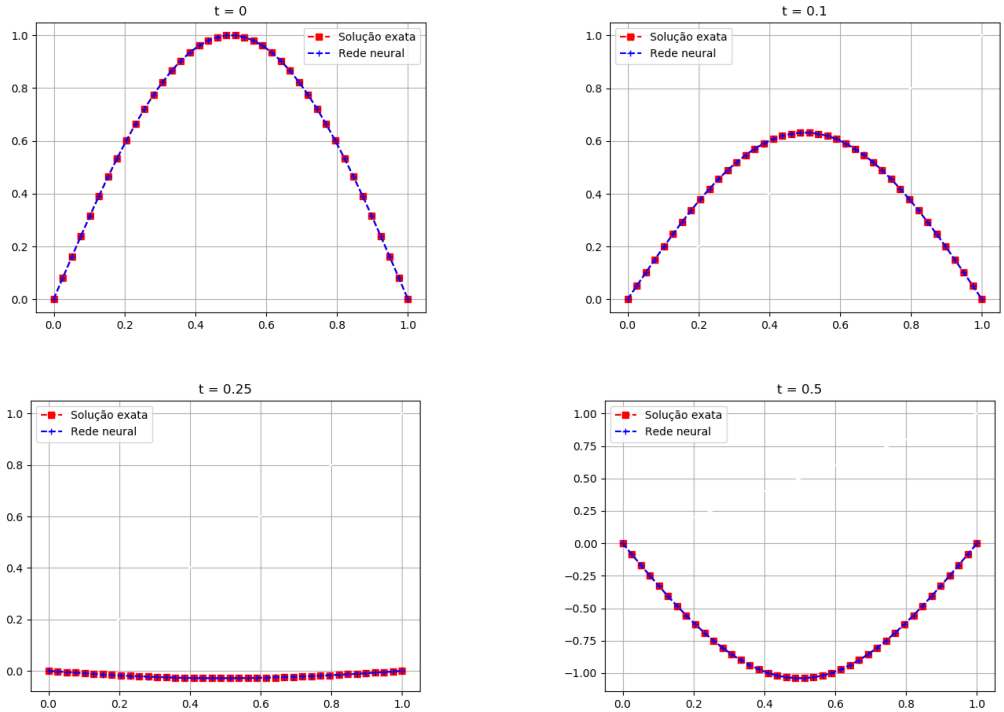


Figura 16 – Soluções exatas e por Redes neurais de 4.15 em diferentes medidas de tempo, com uma camada oculta com 10 neurônios, utilizando uma discretização com $N_x = N_y = 40$.

Este problema foi resolvido usando redes neurais, com 10000 iterações, taxa de aprendizagem $\eta = 10^{-3}$, função de ativação sigmóide e método de minimização Adam, foram

comparados os resultados obtidos com a solução exata (4.19). A Fig. 15 apresenta o resultado das soluções, exata e por redes neurais, enquanto a tabela 8 mostra os erros nas normas do Infinito para várias configurações da rede neurais (camadas ocultas e neurônios por camada). A Fig. 16 faz comparações entre as duas soluções (exata e por RNA) para diferentes medidas de tempo. Nesse experimento estamos considerando $L = \alpha = 1$. Usando 10, 20 e 40 neurônios em até 5 camadas, o erro ficou na ordem de 10^{-3}. O menor erro foi obtido usando 5 camadas ocultas com 40 neurônios cada uma delas.

4.6 Problema unidimensional singularmente pertubado

Nesse experimento vamos estudar o comportamento do método baseado em redes neurais PMC na solução de um problema singularmente pertubado estacionário e unidimensional, o problema de transporte convectivo-difusivo, destacando o caso de convecção dominante. O problema de transporte convectivo-difusivo estacionário 1D consiste em determinar $u : [x_0, x_f] \to \mathbb{R}$, satisfazendo

$$-\epsilon \frac{d^2u(x)}{dx^2} + \beta \frac{du(x)}{dx} = q(x), \quad x \in (x_0, x_f), \tag{4.20}$$

$$u(x_0) = U_0, \tag{4.21}$$

$$u(x_f) = U_f. \tag{4.22}$$

Nesse problema, u representa a concentração de uma substância sendo transportada devido aos processos difusivos (primeiro termo da equação) e convectivos (segundo termo da equação), $\epsilon > 0$ é o coeficiente de difusão, β é a velocidade (constante) e q é uma função dada, o termo de fonte. As condições de contorno de Dirichlet são descritas pelas equações (4.21) e (4.22). Maiores detalhes sobre os aspectos numéricos desse problema pode ser obtido na Referência (QUARTERONI, 2009). De acordo com (QUARTERONI, 2009), considerando $x_0 = 0$, $x_f = 1$, $U_0 = 0$, $U_f = 1$ e $q = 0$, a equação característica associada é

$$-\epsilon\lambda^2 + \beta\lambda = 0,$$

cujas raízes são $\lambda_1 = 0$ e $\lambda_2 = \beta/\epsilon$. A solução geral é da forma

$$u(x) = C_1 e^{\lambda_1 x} + C_2 e^{\lambda_2 x} = C_1 + C_2 e^{\beta x/\epsilon},$$

onde C_1 e C_2 são constantes reais. Usando as condições de contorno (4.21) e (4.22), obtemos as constantes C_1 e C_2, de forma que a solução exata é dada por

$$u(x) = \frac{e^{\frac{\beta x}{\epsilon}} - 1}{e^{\frac{\beta}{\epsilon}} - 1}. \tag{4.23}$$

Vamos avaliar o comportamento da solução exata de acordo com a relação β/ϵ. Por simplicidade, vamos considerar $\beta > 0$. Supondo $\beta/\epsilon << 1$ e usando a expansão de Taylor para funções exponenciais, obtemosdesses

$$u(x) = \frac{1 + \frac{\beta}{\epsilon}x + \cdots - 1}{1 + \frac{\beta}{\epsilon} + \cdots - 1} \approx \frac{\frac{\beta x}{\epsilon}}{\frac{\beta}{\epsilon}} = x,$$

isto é, a solução exata é aproximadamente a reta que interpola as condições de contorno (que é a solução correspondente ao caso $\beta = 0$). Considerando agora $\beta/\epsilon >> 1$, os termos exponenciais assumem valores elevados, de forma que

$$u(x) \approx \frac{e^{\beta x/\epsilon}}{e^{\beta/\epsilon}} = e^{[-\frac{\beta}{\epsilon}(1-x)]}, \tag{4.24}$$

ou seja, a solução é aproximadamente zero em quase todo intervalo $[0, 1]$, exceto na vizinhança do ponto $x = 1$, onde a solução tende a 1 exponencialmente. Essa vizinhança tem um tamanho da ordem de ϵ/β e é, dessa forma, muito pequena. Nessa região, a solução exibe uma *camada limite* de comprimento de ordem $\mathcal{O}(\epsilon/\beta)$ (ver Fig. 17), onde a derivada de u,

$$\frac{du(x)}{dx}\bigg|_{x\approx 1} = \frac{\beta}{\epsilon} e^{[-\frac{\beta}{\epsilon}(1-x)]}\bigg|_{x\approx 1} \approx \frac{\beta}{\epsilon},$$

ou seja, $\frac{du(x)}{dx} \to \infty$ quando $\epsilon \to 0$. A Fig. 17 ilustra os gráficos da solução (4.23) para alguns valores da relação β/ϵ.

A forma usual de medir como o termo convectivo domina o termo difusivo é através do *número de Peclét global*, definido por

$$\mathbb{P}e_g = \frac{|\beta|L}{2\epsilon}, \tag{4.25}$$

onde L é o comprimento característico do domínio espacial ($L = x_f - x_0$ no nosso caso). Como estamos considerando $L = 1$, temos $\frac{|\beta|}{\epsilon} = 2\mathbb{P}e_g$.

A qualidade da solução desse problema obtida por métodos numéricos baseados em malhas, como elementos finitos e diferenças finitas, é caracterizada pelo número adimensional,

$$\mathbb{P}e = \frac{|\beta|h}{2\epsilon}, \tag{4.26}$$

chamado de *número de Péclet local* ou *de malha*, onde h é o parâmetro característico da malha (discretização). Esse número mede a razão entre os efeitos convectivos e os difusivos no contexto da discretização parametrizada por h.

O método de elementos finitos de Galerkin clássico, usando interpolação linear, e o método de diferenças finitas, com diferenças centradas no termo convectivo, apresentam

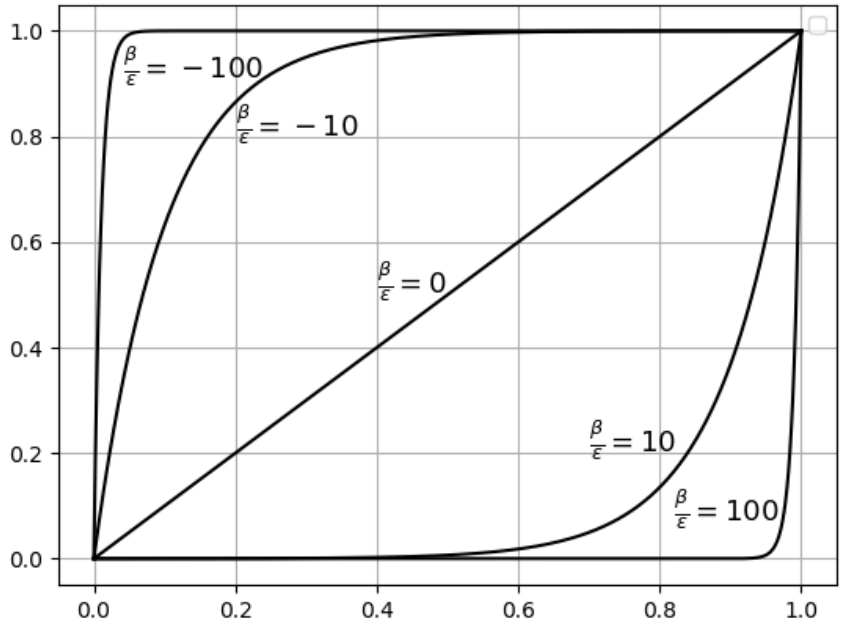


Figura 17 – Comportamento da solução do problema (4.20)-(4.22) em termos da razão $\frac{\beta}{\epsilon}$. Exibimos também os gráficos das soluções para o caso $\beta < 0$.

instabilidades numéricas, conhecidas como oscilações espúrias, quando $\mathbb{P}e > 1$ (ROOS; STYNES; TOBISKA, 1996; QUARTERONI, 2009). Essas oscilações se acentuam nas regiões de camada limite. Uma forma de evitar essas instabilidades numéricas é escolher o comprimento característico h suficientemente pequeno de forma que $\mathbb{P}e < 1$. Contudo, segundo (QUARTERONI, 2009) essa estratégia nem sempre é conveniente, por exemplo, se $\beta = 1$ e $\epsilon = 1/5000$, devemos tomar $h < 10^{-4}$, requerendo uma partição do domínio $[0, 1]$ com no mínimo 10000 subintervalos. Em particular, essa estratégia pode requerer uma quantidade elevada de pontos nodais em problemas descritos em várias dimensões, aumentando consideravelmente o custo computacional. Uma alternativa consiste em usar procedimentos adaptativos que refinam a malha somente nas proximidades da camada limite. Uma outra alternativa é recorrer aos métodos estabilizados (ou métodos de Petrov-Galerkin), no caso de elementos finitos (BROOKS; HUGHES, 1982; HUGHES; FRANCA; HULBERT, 1989; BURMAN; HANSBO, 2004; QUARTERONI, 2009; AUGUSTIN et al., 2011; VALLI et al., 2018), e aos esquemas *upwind* no caso de diferenças finitas (LEVEQUE, 1992; ROOS; STYNES; TOBISKA, 1996; QUARTERONI, 2009). Os métodos de elementos finitos estabilizados estão fora do escopo desse trabalho.

Nosso objetivo nesse experimento numérico é avaliar a performance do método baseado

em redes neurais na solução de problemas de convecção-difusão 1D, especialmente quando $\epsilon << |\beta|$ (ou $Pe >> 1$ para métodos baseados em malha). Vamos comparar os resultados obtidos com os esquemas clássicos de diferenças finitas centrada e atrasada ("*upwind*"), utilizadas para aproximar o termo convectivo da equação. O treinamento da rede será realizado no mesmo conjunto de pontos da discretização utilizada nos métodos de diferenças finitas.

Para discretizar o problema (4.20)-(4.22) pelo método de diferenças finitas, considere uma discretização do domínio $[x_0, x_f]$,

$$x_0 < x_1 < x_2 < \cdots < x_f = x_n,$$

em $n > 1$ subintervalos de tamanhos iguais a $h = x_{i+1} - x_i$, $i = 0, \cdots, n-1$. O termo de segunda ordem (que envolve a derivada segunda), conhecido como termo difusivo, será discretizado via diferenças finitas de segunda ordem, isto é,

$$\frac{d^2u(x_i)}{dx^2} = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} + \mathcal{O}(h^2), \quad i = 1, \cdots, n-1 \tag{4.27}$$

e o termo de primeira ordem (que envolve a derivada primeira), conhecido como termo convectivo ou de transporte, será discretizado utilizando dois esquemas de diferenças finitas: diferenças centradas (segunda ordem) e diferenças atrasadas ("*upwind*- de primeira ordem). O método de diferenças finitas centradas (DFC) no termo convectivo resulta em

$$\frac{du(x_i)}{dx} = \frac{u(x_{i+1}) - u(x_{i-1})}{2h} + \mathcal{O}(h^2), \quad i = 1, \cdots, n-1. \tag{4.28}$$

Descartando o erro de ordem $\mathcal{O}(h^2)$ e fazendo $u_i \approx u(x_i)$, obtemos o primeiro esquema de diferenças finitas,

$$-\epsilon\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \beta\frac{u_{i+1} - u_{i-1}}{2h} = q_i, \quad i = 1, \cdots, n-1, \tag{4.29}$$

onde $q_i = q(x_i)$, $u_0 = U_0$ e $u_n = U_f$ são dados pelas condições de contorno. Para cada i, a variável u_i fornece uma aproximação para o valor exato $u(x_i)$, $x_i = x_0 + ih$. A expressão (4.29) pode ser reescrita da forma

$$(1 + \mathbb{P}e)u_{i-1} - 2u_i + (1 - \mathbb{P}e)u_{i+1} = -\frac{h^2 q_i}{\epsilon}, \quad i = 1, , n-1,$$

com $\mathbb{P}e$ dado por (4.26), que resulta no sistema linear tridiagonal

$$\begin{bmatrix} -2 & (1-\mathbb{P}e) & 0 & 0 & 0 & \cdots & 0 \\ (1+\mathbb{P}e) & -2 & (1-\mathbb{P}e) & 0 & 0 & \cdots & 0 \\ 0 & (1+\mathbb{P}e) & -2 & (1-\mathbb{P}e) & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & & \cdots & \vdots \\ 0 & \cdots & & & 0 & (1+\mathbb{P}e) & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \end{bmatrix} = -\frac{h^2}{\epsilon} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{n-1} \end{bmatrix} + \boldsymbol{r},$$

onde $\boldsymbol{r} = \Big(-(1+\mathbb{P}e)U_0, 0, \cdots, 0, -(1-\mathbb{P}e)U_f \Big)^T$ é um vetor de ordem $n-1$ associado às condições de contorno de Dirichlet. A solução desse sistema fornece as soluções $u_1, \cdots, u_{n-1}$ nos pontos $x_1, \cdots, x_{n-1}$ do método DFC. Como veremos nos experimentos numéricos, o esquema numérico (4.29) obtido usando o método DFC no termo convectivo apresenta oscilações espúrias quando $\mathbb{P}e > 1$. O método de elementos finitos de Galerkin clássico quando aplicado ao problema (4.20)-(4.22) é equivalente ao esquema (4.29), apresentando as mesmas oscilações espúrias (ROOS; STYNES; TOBISKA, 1996; QUARTERONI, 2009), por isso não o descrevemos aqui.

O método de diferenças finitas atrasadas discretiza o termo convectivo no ponto x_i utilizando o valor da solução em x_{i-1} se $\beta > 0$ e o valor da solução em x_{i+1} se $\beta < 0$. Considerando o caso $\beta > 0$, temos

$$\frac{du(x_i)}{dx} = \frac{u(x_i) - u(x_{i-1})}{h} + \mathcal{O}(h), \quad i = 0, \cdots, n. \tag{4.30}$$

O esquema de diferenças finitas resultante de (4.30) e (4.27), conhecido como esquema *upwind*, é dado por

$$-\epsilon \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \beta \frac{u_i - u_{i-1}}{h} = q_i, \quad i = 0. \cdots, n-1, \tag{4.31}$$

Essa última equação pode ser reescrita da forma

$$(1+2\mathbb{P}e)u_{i-1} - 2(1+\mathbb{P}e)u_i + u_{i+1} = -\frac{h^2 q_i}{\epsilon}, \quad i = 1, , n-1,$$

que resulta no sistema linear tridiagonal

$$\begin{bmatrix} -2(1+\mathbb{P}e) & 1 & 0 & 0 & \cdots & 0 \\ (1+2\mathbb{P}e) & -2(1+\mathbb{P}e) & 1 & 0 & \cdots & 0 \\ 0 & (1+2\mathbb{P}e) & -2(1+\mathbb{P}e) & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ 0 & \cdots & & 0 & (1+2\mathbb{P}e) & -2(1+\mathbb{P}e) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \end{bmatrix} = -\frac{h^2}{\epsilon} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{n-1} \end{bmatrix} + \boldsymbol{r},$$

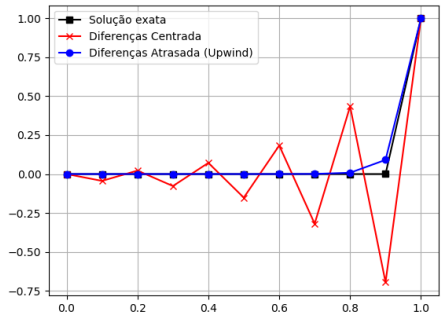
onde $\boldsymbol{r} = \Big((1+2\mathbb{P}e)U_0, 0, \cdots, 0, -U_f \Big)^T$ é um vetor de ordem $n-1$ associado às condições de contorno de Dirichlet. A solução desse sistema fornece os valores $u_1, \cdots, u_{n-1}$ nos pontos $x_1, \cdots, x_{n-1}$. O esquema numérico *upwind* apresenta soluções estáveis para problemas predominantemente convectivos, isto é, quando $\mathbb{P}e > 1$. Por outro lado, a ordem de convergência desse método é $\mathcal{O}(h)$ (linear), enquanto que o método DFC possui ordem de convergência $\mathcal{O}(h^2)$ (quadrática).

A Figura 18 apresenta as soluções do problema (4.20)-(4.22), considerando $x_0 = 0$, $x_f = 1$, $U_0 = 0$, $U_f = 1$ e $q = 0$, com os métodos de diferenças finitas centrada e atrasada (*upwind*), usando número de Péclet global $\mathbb{P}e_g = 50$ e vários valores do número de Péclet local. Nas figura (a) e (b) observamos o comportamento desses métodos quando $\mathbb{P}e > 1$: em (a), para $\mathbb{P}e = 5$ o método de diferenças centrada é totalmente oscilatório, enquanto o método *upwind* é livre de oscilações; em (b), para $\mathbb{P}e = 1,25$ (ligeiramente maior do que 1), o método de diferenças centrada apresenta uma oscilação exatamente na camada limite próximo a $x = 1$, enquanto o método *upwind* é livre de oscilações, mas não é muito preciso. Na figura (c), como $\mathbb{P}e \approx 0,7 < 1$, os métodos de diferenças centrada e *upwind* são estáveis, mas o método de diferenças centrada é mais preciso, devido sua convergência ser quadrática.

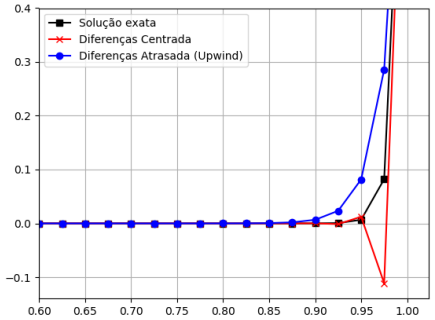
A forma geral da solução aproximada do problema (4.20)-(4.22) via redes neurais é dada por

$$u_{RN}(x) = u_{RN}(x, \boldsymbol{p}) = \Big(\frac{x_f - x}{x_f - x_0}\Big)U_0 + \Big(\frac{x - x_0}{x_f - x_0}\Big)U_f + (x - x_0)(x - x_f)N(x, \boldsymbol{p}) \tag{4.32}$$

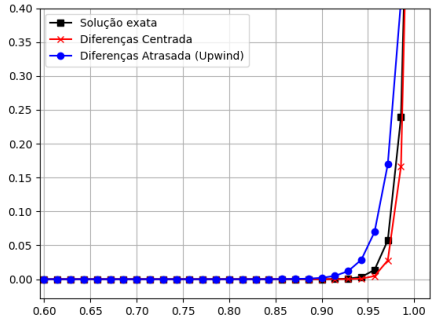
$$= x + x(x-1)N(x, \boldsymbol{p}), \tag{4.33}$$



(a) $\mathbb{P}e = 5$



(b) $\mathbb{P}e = 1,25,\ 0,6 \leq x \leq 1$ e $-1,5 \leq y \leq 0,4$



(c) $\mathbb{P}e \approx 0,7,\ 0,6 \leq x \leq 1$ e $0 \leq y \leq 0,4$

Figura 18 – Soluções do problema (4.20)-(4.22) via métodos de diferenças finitas (centrada e atrasada) com $\mathbb{P}e_g = 50$ e vários valores do número de Péclet local.

de forma que o funcional resíduo discreto a ser minimizado é dado por

$$J_R(\boldsymbol{p}) = \sum_{x_j \in \mathcal{P}_x} \Big[-\epsilon \frac{d^2}{dx^2} u_{RN}(x_j, \boldsymbol{p}) + \beta \frac{d}{dx} u_{RN}(x_j, \boldsymbol{p}) - q(x_j) \Big]^2, \tag{4.34}$$

onde $\mathcal{P}_x$ é uma discretização do domínio espacial $[0, 1]$. Utilizaremos a mesma discretização dos métodos de diferenças finitas. Além disso, usaremos o algoritmo de otimização *Adam* da biblioteca *TensorFlow* para minimizar o funcional (4.34).

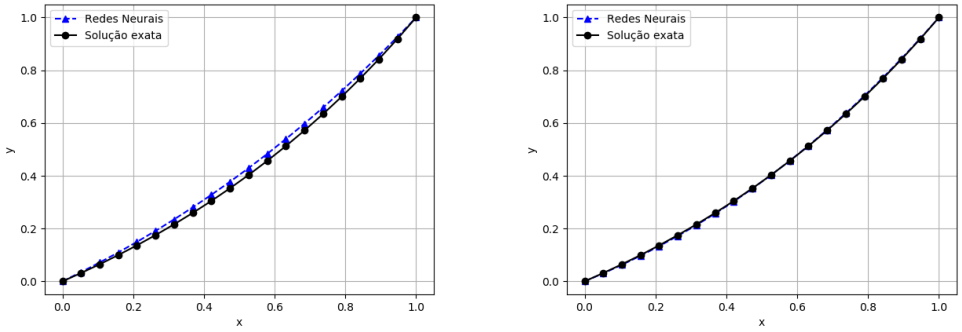
Testamos o método baseado em redes neurais na solução desse problema considerando vários valores dos números de Péclet global e local. Vale ressaltar que o número de Péclet local é definido para métodos baseados em malhas, que não o caso do método de redes neurais, que não precisa necessariamente de uma malha, mas sim de um conjunto de pontos quaisquer. Como as redes neurais utilizadas serão treinadas usando os mesmos pontos da malha discretizada para os métodos de diferenças finitas, utilizaremos também o número de Péclet local para estudar o comportamento da solução obtida via redes neurais. Começamos com um problema com os seguintes dados

$$\epsilon = \beta = 1, \quad \mathbb{P}e_g = 0.5,$$

de forma que $\mathbb{P}e = 0.026$, ou seja, um problema com difusão dominante ($\mathbb{P}e < 1$) e sem camadas limites. A Fig. 19 mostra os resultados obtidos usando uma rede neural com uma única camada oculta com 20 neurônios e um conjunto de treinamento com 20 pontos (equidistantes) nodais. A figura (a) mostra o gráfico da solução

considerando o treinamento com apenas 50 iterações, enquanto que o gráfico da figura (b) foi obtido com o treinamento com 100 iterações. Visualmente, as soluções exata e aproximada coincidem, considerando 100 iterações. Neste caso, o erro na norma do infinito foi de $\|u - u_{RN}\|_\infty = 3,204 \times 10^{-3}$, enquanto que na figura (a) esse erro foi de $\|u - u_{RN}\|_\infty = 2,660 \times 10^{-2}$.

A Fig. 20 apresenta as soluções obtidas por todos os três métodos (Diferenças Centrada, Diferenças Atrasada (*upwind*) e Redes Neurais) e seus erros usando a norma do infinito. O erro do método de Diferenças Centrada, $\|u - u_{DC}\|_\infty = 2,790 \times 10^{-5}$, foi o menor. Nessa configuração, a rede neural considerada foi formada por duas camadas ocultas com 100 neurônios cada, com um treinamento de 2000 iterações, mas mesmo assim o erro obtido não decresceu consideravalmente, comparado com o resultado obtido na Fig. 19-(b). Nesse experimento, o erro obtido pela rede neural foi de $\|u - u_{RN}\|_\infty = 2,828 \times 10^{-3}$, enquanto que o método de Diferenças Atrasada obteve erro levemente inferior, $\|u - u_{DA}\|_\infty = 3,075 \times 10^{-3}$. Mesmo considerando 100 pontos nodais, duas camadas ocultas com 100 neurônios cada, com um treinamento de 2000 iterações, o erro da solução obtida via redes neurais é $\|u - u_{RN}\|_\infty = 2,441 \times 10^{-3}$, e com treinamento de 5000 iterações, o erro é $\|u - u_{RN}\|_\infty = 1,729 \times 10^{-3}$.



(a) Treinamento com 50 iterações
$\|u - u_{RN}\|_\infty = 0,0266$

(b) Treinamento com 100 iterações
$\|u - u_{RN}\|_\infty = 0,0032$

Figura 19 – Soluções do problema (4.20)-(4.22) via redes neurais com $\mathbb{P}e_g = 0.5$ e $\mathbb{P}e = 0.026 < 1$. A rede neural contém uma única camada oculta com 20 neurônios.

Vamos avaliar agora o problema (4.20)-(4.22) com os seguintes dados

$$\epsilon = 10^{-1}, \quad \beta = 1, \quad \mathbb{P}e_g = 5.$$

Primeiro, vamos considerar uma malha de treinamento da rede com 20 pontos nodais, isto é, $\mathbb{P}e = 0.26$. Neste caso, o problema é considerado difusão dominante ($\mathbb{P}e < 1$). O método baseado em redes neurais teve uma certa dificuldade para resolver o problema com esses coeficientes, como pode ser visto na Fig. 21. Considerando uma única camada oculta com 100 neurônios e número de iterações (treinamento) igual até 5000 (figuras (a) e (b)), o método não foi capaz de resolver o problema satisfatoriamente. Mesmo com duas camadas ocultas, com 100 neurônios cada uma delas e treinamento da rede com 1000 iterações, a solução obtida ficou ruim, conforme figura (c). Por outro lado, treinando a rede com duas comadas ocultas e com 1500 iterações, a solução obtida coincidiu com a solução exata, com erro $\|u - u_{RN}\|_\infty = 9,818 \times 10^{-4}$ (ver figura (d)).

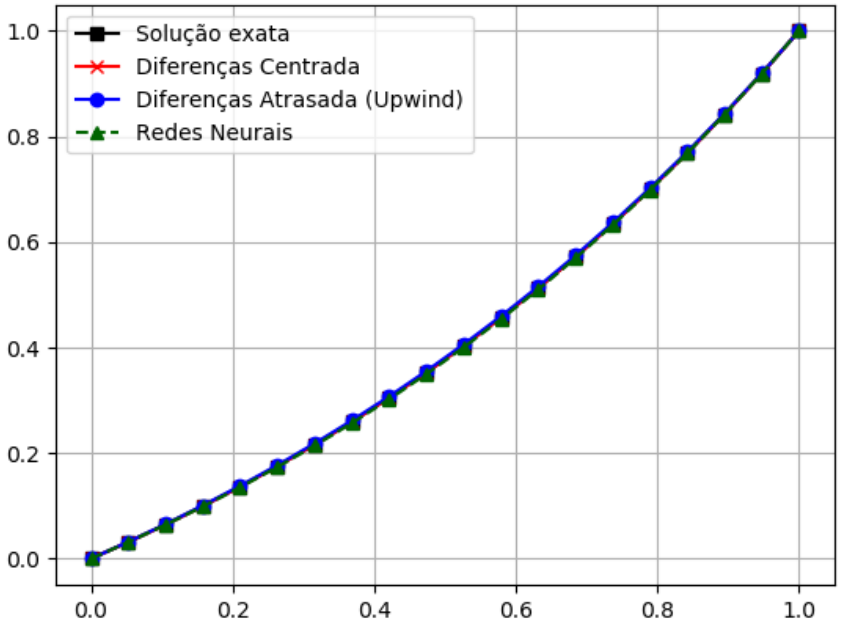


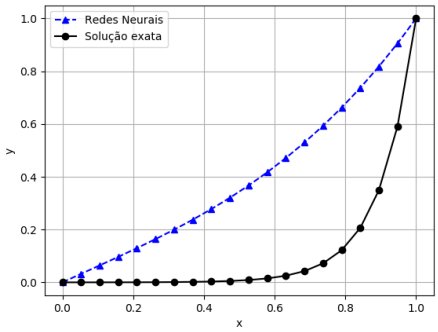
Figura 20 – Soluções do problema (4.20)-(4.22) com $\mathbb{P}e_g = 0.5$ e $\mathbb{P}e = 0.026 < 1$ em uma malha com 20 pontos nodais e erros na norma do infinito: $\|u - u_{DC}\|_\infty = 2,790 \times 10^{-5}$, $\|u - u_{DA}\|_\infty = 3,075 \times 10^{-3}$, $\|u - u_{RN}\|_\infty = 2,828 \times 10^{-3}$. A rede neural contém duas camadas ocultas com 100 neurônios cada e o treinamento foi realizado com 2000 iterações.

A Fig. 22 apresenta as soluções obtidas por todos os três métodos (Diferenças Centrada, Diferenças Atrasada (*upwind*) e Redes Neurais) e seus erros usando a norma do infinito. O método baseado em redes neuras obteve o menor erro, e consequentemente, a melhor solução, apesar de visualmente, essa solução coincidir com a solução obtida pelo método de Diferenças Centrada. Vale ressaltar que essa solução é a mesma apresentada na Fig. 21-(d). O método de Diferenças Atrasada teve um comportamento mais difusivo, obtendo o pior resultado.

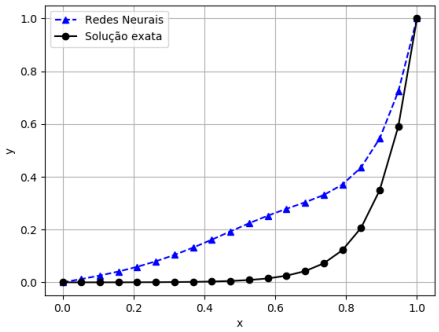
Finalmente, vamos estudar um caso com camada limite acentuada (e convecção dominante), onde

$$\beta = 1, \quad \mathbb{P}e_g = 50.$$

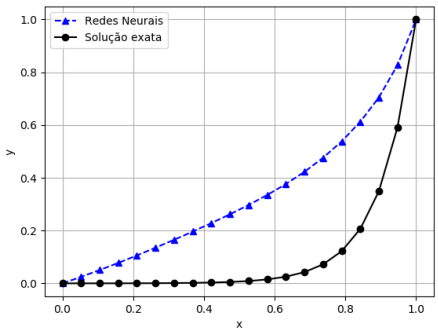
Para que tenhamos um número de Péclet local $\mathbb{P}e = 5$, escolhemos $\epsilon = 10^{-2}$ e uma discretização do domínio com 11 pontos nodais. Os resultados obtidos pelos métodos de Diferenças Centrada e Diferenças Atrasada são mostrados na Fig. 18-(a). A Fig. 23 mostra os resultados usando o método baseado em redes neurais, com uma única camada oculta composta por 100 neurônios e treinamento com 1000 iterações (figura (a)) e com duas camadas ocultas compostas por 100 neurônios cada uma delas e treinamento com 5000 iterações (figura (b)). Essas soluções não tem nada a ver com a solução exata, indicando que o método não é capaz de resolver esse problema. No caso de uma única camada oculta, fizemos testes aumentando consideravelmente o número de neurônios, assim como o número de pontos nodais do domínio (diminuindo o $\mathbb{P}e$) e o número de iterações, mas as soluções obtidas continuam próximas da solução da figura (a). Interpretamos o caso da



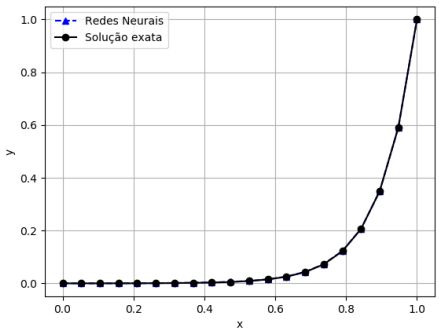
(a) Uma camada oculta com 100 neurônios, treinamento com 1000 iterações, $\|u - u_{RN}\|_\infty = 5,404 \times 10^{-1}$.



(b) Uma camada oculta com 100 neurônios, treinamento com 5000 iterações, $\|u - u_{RN}\|_\infty = 2,607 \times 10^{-1}$.



(c) Duas camada ocultas com 100 neurônios cada, treinamento com 1000 iterações, $\|u - u_{RN}\|_\infty = 4,161 \times 10^{-1}$.



(d) Duas camada ocultas com 100 neurônios cada, treinamento com 1500 iterações, $\|u - u_{RN}\|_\infty = 9,818 \times 10^{-4}$.

Figura 21 – Soluções do problema (4.20)-(4.22) via redes neurais com $\mathbb{P}e_g = 5$ e $\mathbb{P}e = 0.263 < 1$ (20 pontos nodais).

figura (b) como se o método baseado em redes neurais "convergisse"para uma aproximação da função

$$s(x) = \begin{cases} 1, & x \in (0, 1], \\ 0, & x = 0, \end{cases} \tag{4.35}$$

que também satisfaz o valor mínimo do funcional (4.34), quando $\mathbb{P}e_g \to \infty$ (ou $\frac{\beta}{\epsilon} = 2\mathbb{P}e_g \to \infty$), ou seja, qualquer função u_{RN} que seja constante *quase sempre* em $(0, 1)$, satisfazendo as condições de contorno, é solução do problema de minimização do funcional resíduo. Para este caso, fizemos testes aumentando o número de camadas ocultas (até 7 camadas), o número de pontos nodais e o número de iterações, mas a solução aproximada não melhora.

Percebemos através desse experimento que o método baseado em redes neurais PMC também possui certa dificuldade na solução de problemas predominantemente convectivos. Na literatura, os únicos trabalhos encontrados envolvendo a solução de problemas predominantemente convectivos são (LI et al., 2010) e (SILVA; NEITZEL; LIMA, 2008). Em (SILVA; NEITZEL; LIMA, 2008), os autores utilizam uma junção de redes neurais PMC com funções-base radiais (RBF) para estudar dois problemas de convecção-difusão unidimensional com

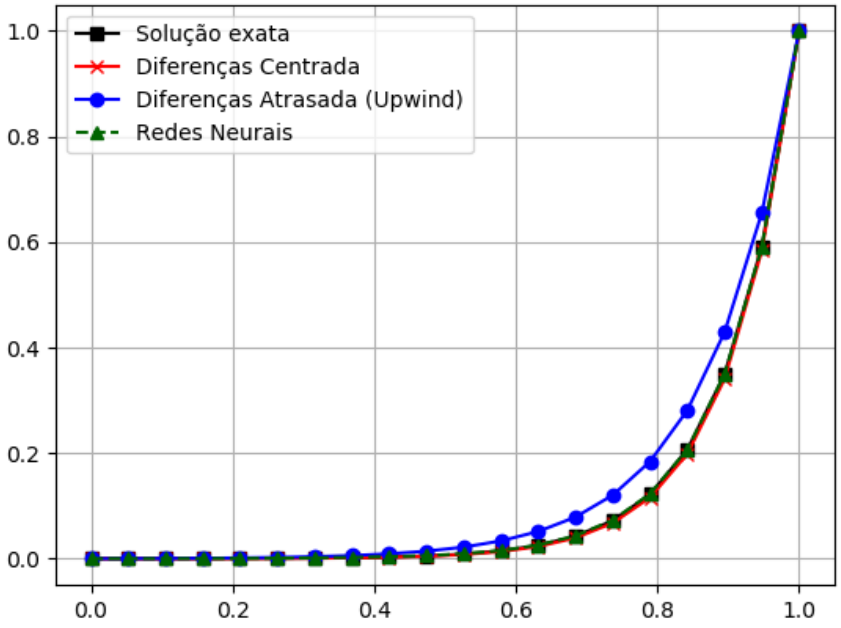
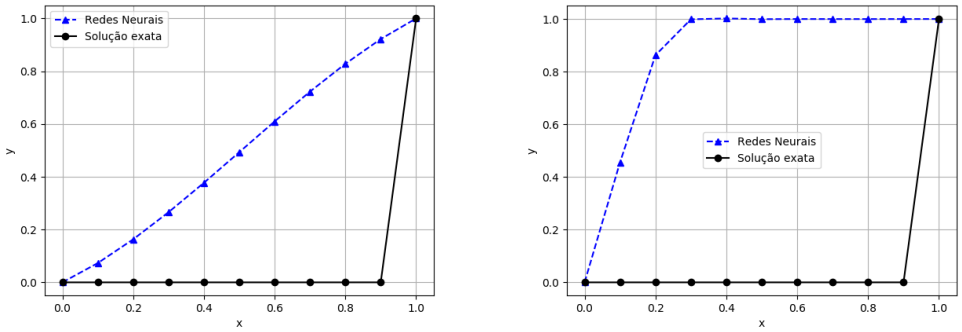


Figura 22 – Soluções do problema (4.20)-(4.22) com $\mathbb{P}e_g = 5$ e $\mathbb{P}e = 0.102 < 1$ em uma malha com 20 pontos nodais e erros na norma do infinito: $\|u - u_{DC}\|_\infty = 8,734 \times 10^{-3}$, $\|u - u_{DA}\|_\infty = 8,008 \times 10^{-2}$, $\|u - u_{RN}\|_\infty = 9,818 \times 10^{-4}$. A rede neural contém duas camadas ocultas com 100 neurônios cada e o treinamento foi realizado com 1500 iterações.

soluções apresentando gradientes elevados (camadas limites). Em (LI et al., 2010) os autores propõem o método *Integration Wavelet Neural Network* (I-WNN) baseado em funções *spline wavelet* para resolver equações diferenciais. Eles aplicam a metodologia proposta em dois problemas de convecção-difusão unidimensional com convecção dominante e concluem que o método representa bem a solução para problemas com $\frac{\beta}{\epsilon} = 100$ (razão entre convecção e difusão). Nos métodos de redes neurais clássicos para resolver equações diferenciais, a não linearidade é aproximada, em geral, por superposições da função de ativação sigmóide, enquanto que no método *Wavelet Neural Network* (WNN), a não linearidade é aproximada por superposições de uma série de funções *wavelet*. O método I-WNN é um método WNN que utiliza um processo de integração ao invés de diferenciação para resolver a equação diferencial. A função de ativação do método I-WNN, obtida através de funções *spline wavelets*, é descrita por

$$\varphi(x) = \frac{1}{6}\begin{cases} 0, & x \leq 0, \\ x^3, & x \in [0,1], \\ 4 - 12x + 12x^2 - 3x^3, & x \in [1,2], \\ -44 + 60x - 24x\ 2 + 3x^3, & x \in [2,3], \\ 64 - 48x + 12x^2 - x^3, & x \in [3,4], \\ 0, & x \geq 4. \end{cases}$$



(a) Uma camada oculta com 100 neurônios, treinamento com 1000 iterações.

(b) Duas camadas ocultas com 100 neurônios cada, treinamento com 5000 iterações.

Figura 23 – Soluções do problema (4.20)-(4.22) via redes neurais com $\mathbb{P}e_g = 50$ e $\mathbb{P}e = 5$ (11 pontos nodais).

Um dos experimentos testados é dado pelo problema (4.20)-(4.22) com $\beta = 1$, $q = 1$, $U_0 = U_f = 1$ e ϵ é escolhido de forma que $\frac{\beta}{\epsilon} = 2\mathbb{P}e_g = 10, 40, 100$. A solução exata é dada por

$$u(x) = x - \frac{1 - e^{\beta x/\epsilon}}{1 - e^{\beta/\epsilon}}.$$

Os autores não informam no artigo o número de pontos de discretização do domínio (visualmente parece ter 21 pontos nodais), número de camadas ocultas, números de neurônios e nem a quantidade de iterações do treinamento da rede. Resolvemos esse problema para o pior caso, $\frac{\beta}{\epsilon} = 100$, usando a metodologia estudada nessa dissertação (com função de ativação sigmoidal). O melhor resultado foi obtido usando uma discretização com 101 pontos nodais, 3 camadas ocultas com 100 neurônios cada uma delas e um treinamento com 10000 (dez mil) iterações. A Fig. mostra a solução obtida com erro na norma do infinito $\|u - u_{RN}\|_\infty = 6,165 \times 10^{-3}$. Usando configurações inferiores a essa, o método não foi capaz de resolver o problema satisfatoriamente.

Mais recentemente, Liu et al. (2020) apresentam um método basedo em redes neurais com polinômios de Legendre (*Legendre Neural Network Method*) para algumas classes de problemas singularmente pertubados.

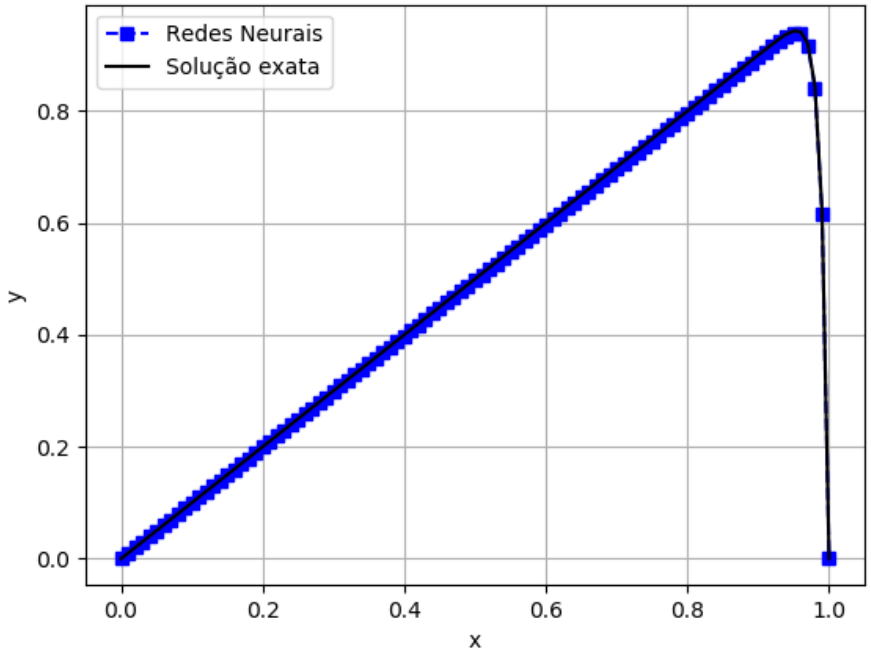


Figura 24 – Solução do problema descrito em (LI et al., 2010) com $\mathbb{P}e_g = 50$ em uma malha com 101 pontos nodais e uma rede neural com três camadas ocultas com 100 neurônios cada e treinamento com 10000 iterações. O erro na norma do infinito é $\|u - u_{RN}\|_\infty = 6,165 \times 10^{-3}$.

5 Considerações Finais e Recomendações

O estudo de equações diferenciais exerce um importante papel em vários campos da ciênciae tecnologia, através da modelagem de problemas do mundo real. Como grande parte dos modelos matemáticos descritos por equações diferenciais (ordinárias e parciais) não possui solução analítica, método numéricos, como diferenças finitas e elementos finitos, são utilizados para resolvê-lo. Recentemente, muitos estudos têm sido dedicados na aplicação de Redes Neurais Artificiais (RNA) no processo de solução de equações diferenciais, com resultados promissores. No entanto, a confiabilidade e a precisão da aproximação ainda representam questões importantes que não estão totalmente resolvidas na literatura atual. As metodologias computacionais, em geral, dependem de uma variedade de parâmetros, bem como da escolha dos métodos de otimização, um ponto que deve ser visto junto com a estrutura da função custo (funcional resíduo).

Nesta dissertação foi apresentada um estudo do processo de solução de equações diferenciais ordinárias e parciais, usando redes neurais *feedforward*. Este trabalho foi desenvolvido na linguagem Python com a utilização do Tensorflow, o que proporcionou algumas facilidades, tal como a utilização de funções de ativação e métodos de minimização previamente implementados no framework. Em contrapartida, não é trivial desenvolver por meio deste uma função de ativação genérica, tampouco, implementar métodos de minimização genéricos.

As redes neurais utilizadas nos experimentos numéricos obtiveram um bom desempenho computacional, tanto em termos de tempo e processamento. Foram utilizadas redes neurais com até 5 camadas ocultas e 500 neurônios, além de um processo de treinamento com 100000 iterações.

Foram executados diversos testes com alterações no valor da taxa de aprendizagem em todos os problemas apresentados neste trabalho e o valor de 10^{-3} fez-se a opção com melhor resultado possível. Também foi percebido que quanto maior era o número de pontos na discretização, seria necessário aumentar a quantidade de neurônios em cada camada oculta a fim de se alcançar os melhores resultados, confome observado na seção 4.

Em concordância do que foi verificado através das pesquisas decorrentes deste trabalho, a metologia aplicada ainda é extensamente utilizada em boa parte dos trabalhos desenvolvidos, todavia, a função da solução aproximada para o cálculo do funcional resíduo é diferente para cada tipo de problema, dado que as condições de contorno são inseridas na solução aproximada.

Como sugestões para trabalhos futuros, destacamos:

- Solução de problemas não lineares;
- Soluções de problemas vetoriais, como as equações de Stokes, Navier-Stokes, etc;
- Soluções de problemas com muitas variáiveis independentes, como àqueles que aparecem na área de finanças;
- Inclusão das condições de contorno no funcional resíduo;
- Implementação computacional utilizando outras bibliotecas, como o *Keras*, *Theano* e o *Torch*, amplamente difundidos na área de aprendizagem profunda;
- Utilização de outras arquiteturas de redes neurais (redes convolucionais, redes recorrentes, etc.);

- Implementação de métodos de minimização e funções de ativação diferentes das disponíveis na biblioteca do Tensorflow;
- Desenvolvimento e estudos de métodos para domínios mais gerais (não retangulares);
- Treinamento da rede utilizando pontos aleatórios do domínio;
- Desenvolvimento de um estudo avaliando a variação de todos os parâmetros envolvidos na rede neural e suas dependências mutuas, por exemplo: métodos de inicialização do pesos e limiares, número de camadas ocultas, número de neurônios por camadas, treinamento da rede, taxa de aprendizagem variável, métodos de otimização, etc.
- Desenvolvimento de métodos baseado em redes neuras para problemas predominantemente convectivos.

Referências

AARTS, L. P.; VEER, P. van der. Neural network method for solving partial differential equations. Neural Processing Letters, v. 14, p. 261–271, 2001. Citado na página 25.

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. In: . USA: USENIX Association, 2016. (OSDI'16), p. 265–283. ISBN 9781931971331. Citado 2 vezes nas páginas 27 e 62.

ABRAHAMS, S. et al. TensorFlow for Machine Intelligence: A Hands-On Introduction to Learning Algorithms. Bleeding Edge Press, 2016. ISBN 978-1-939902-35-1. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=7C4333E23A6BF64508F7C848E13CE5FB>. Citado 2 vezes nas páginas 27 e 62.

AGGARWAL, C. C. Neural Networks and Deep Learning. A Textbook. [S.l.]: Springer, 2018. ISBN 978-3-319-94463-0. Citado na página 26.

AUGUSTIN, M. et al. An assessment of discretizations for convection-dominated convection–diffusion equations. Comput. Methods Appl. Mech. and Engrg., v. 200, p. 3395–3409, 2011. Citado na página 78.

BAR, L.; SOCHEN, N. Unsupervised Deep Learning Algorithm for PDE-based Forward and Inverse Problems. arXiv:1904.05417, 2019. Citado na página 51.

BAYAMANI, M.; EFFATI, S.; KERAYECHIAN, A. A feed-forward neural network for solving stokes problem. Acta Appl Math, v. 116, p. 55–64, 2011. Citado na página 25.

BAYDIN, A. G. et al. Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research, v. 18, n. 153, p. 1–43, 2018. Disponível em: <http://jmlr.org/papers/v18/17-468.html>. Citado 2 vezes nas páginas 27 e 50.

BECK, C. et al. Solving stochastic differential equations and Kolmogorov equations by means of deep learning. 2018. Citado na página 26.

BEIDOKHTI, R. Shekari; MALEK, A. Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques. Journal of the Franklin Institute, v. 346, n. 9, p. 898–913, 2009. ISSN 0016-0032. Citado 4 vezes nas páginas 25, 52, 99 e 100.

BELLMAN, R. Dynamic Programming. [S.l.]: Princeton University Press, 1957. Citado na página 23.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council, v. 5, p. 157–66, 02 1994. Citado na página 24.

BOTTOU, L.; CURTIS, E. F.; NOCEDAL, J. Optimization methods for large-scale machine learning. SIAM Review, v. 60, n. 2, p. 223–311, 2018. Citado na página 50.

BRAGA, A. de P.; CARVALHO, A. de L. F.; LUDERMIR, T. Redes neurais artificiais: teoria e aplicações. [S.l.]: LTC Editora, 2000. ISBN 9788521612186. Citado 3 vezes nas páginas 32, 34 e 38.

BRAUN, M. Book. Differential equations and their applications : an introduction to applied mathematics / M. Braun. [S.l.]: Springer-Verlag New York, 1975. xiv, 718 p. : p. ISBN 0387901140. Citado na página 23.

BROOKS, A.; HUGHES, T. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. v. 32, p. 199–259, 1982. Citado na página 78.

BURMAN, E.; HANSBO, P. Edge stabilization for galerkin approximations of convection–diffusion–reaction problems. Comput. Methods Appl. Mech. and Engrg., v. 193, n. 15, p. 1437 – 1453, 2004. Citado na página 78.

CHAUDHARI, P. et al. Deep Relaxation: partial differential equations for optimizing deep neural networks. 2017. Citado na página 26.

CHOLLET, F. et al. Keras. 2015. <https://keras.io>. Citado na página 27.

COLLOBERT, R.; BENGIO, S.; MARITHOZ, J. Torch: A modular machine learning software library. 11 2002. Citado na página 27.

CURRY, H. B. The method of steepest descent for non-linear minimization problems. Quart. Appl. Math., v. 2, p. 258–261, 1944. Citado na página 50.

CYBENKO, G. Continuous valued neural networks with two hidden layers are sufficient. [S.l.]: University of Illinois at Urbana-Champaign. Center for Supercomputing Research and Development, 1988. Citado na página 36.

CYBENKO, G. Approximation by superposition of a sigmoidal function. Math. Control Signals Systems, v. 2, p. 303–314, 1989. Citado 3 vezes nas páginas 25, 43 e 44.

DISSANAYAKE, M. W. M. G.; PHAN-THIEN, N. Neural-network-based approximations for solving partial differential equations. Communications in Numerical Methods in Engineering, v. 10, p. 195–201, 1994. Citado na página 50.

DOCKHORN, T. A discussion on solving partial differential equations using neural networks. arXiv:1904.07200, 2019. Citado 2 vezes nas páginas 26 e 51.

E, W.; HAN, J.; JENTZEN, A. Algorithms for solving high dimensional pdes: From nonlinear monte carlo to machine learning. CoRR, abs/2008.13333, 2020. Disponível em: <https://arxiv.org/abs/2008.13333>. Citado na página 23.

GARDNER, M.; DORLING, S. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric Environment, v. 32, n. 14, p. 2627 – 2636, 1998. ISSN 1352-2310. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1352231097004470>. Citado 2 vezes nas páginas 35 e 36.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.). Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 249–256. Citado 2 vezes nas páginas 33 e 63.

GRAVE, M. et al. Assessing the spatio-temporal spread of COVID-19 via compartmental models with diffusion in Italy, USA, and Brazil. 2021. Citado na página 23.

GRIEWANK, A.; WALTHER, A. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. 2 ed.. ed. Society for Industrial and Applied Mathematics, 2008. Disponível em: <https://epubs.siam.org/doi/abs/10.1137/1.9780898717761>. Citado na página 27.

GUO, Q.; LIU, J.-G.; WANG, D.-H. A modified bfgs method and its superlinear convergence in nonconvex minimization with general line search rule. Journal of Applied Mathematics and Computing, v. 28, p. 435–446, 08 2008. Citado na página 50.

GUO, Y. et al. Solving partial differential equations using deep learning and physical constraints. Applied Sciences, v. 10, p. 5917, 08 2020. Citado na página 26.

HAGAN, M. T.; MENHAJ, M. B. Training feedforward networks with the marquardt algorithm. IEEE Transactions on Neural Networks, v. 5, n. 6, p. 989–993, 1994. Citado na página 50.

HAN, J.; JENTZEN, A.; WEINAN, E. Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning. ArXiv, abs/1707.02568, 2017. Citado na página 26.

HANIN, B.; SELLKE, M. Approximating Continuous Functions by ReLU Nets of Minimal Width. 2018. Citado na página 43.

HAYKIN, S. Redes Neurais: Princípios e Prática. [S.l.]: Artmed, 2007. ISBN 9788577800865. Citado 4 vezes nas páginas 29, 31, 37 e 38.

HORNIK, K. Approximation capabilities of multilayer feedforward networks. Neural Networks, v. 4, n. 2, p. 251 – 257, 1991. ISSN 0893-6080. Citado na página 43.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. Neural Networks, v. 2, n. 5, p. 359 – 366, 1989. ISSN 0893-6080. Citado 2 vezes nas páginas 25 e 43.

HUGHES, T. The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. Prentice-Hall, 1987. ISBN 9780133170252. Disponível em: <https://books.google.com.br/books?id=pF-IQgAACAAJ>. Citado na página 26.

HUGHES, T.; FRANCA, L.; HULBERT, G. A new finite element formulation for computational fluid dynamics: VIII. the Galerkin/least-squares method for advective-diffusive equations. Comput. Methods Appl. Mech. and Engrg., v. 73, p. 173–189, 1989. Citado na página 78.

JAIN, A. K.; MAO, J.; MOHIUDDIN, K. M. Artificial neural networks: A tutorial. Computer, IEEE Computer Society Press, v. 29, n. 3, p. 31–44, mar. 1996. Citado na página 29.

JENTZEN, M. H. abd A.; KRUSE, T.; NGUYEN, T. A. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. SN Partial Differ. Equ. Appl., v. 1-10, 2020. Citado na página 26.

JIANYU, L. et al. Numerical solution of elliptic partial differential equation using radial basis function neural networks. Neural Networks, v. 16, n. 5, p. 729–734, 2003. ISSN 0893-6080. Citado na página 25.

KIDGER, P.; LYONS, T. Universal Approximation with Deep Narrow Networks. 2020. Citado na página 43.

KINGMA, D. P.; BA, J. Adam: A Method for Stochastic Optimization. 2017. Citado na página 50.

KRATSIOS, A. The universal approximation property: Characterization, construction, representation, and existence. Annals of Mathematics and Artificial Intelligence, 2021. Citado na página 43.

KUMAR, M.; YADAV, N. Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey. Computers Mathematics with Applications, v. 62, n. 10, p. 3796–3811, 2011. ISSN 0898-1221. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0898122111007966>. Citado na página 25.

KUMAR, S. K. On weight initialization in deep neural networks. CoRR, abs/1704.08863, 2017. Disponível em: <http://arxiv.org/abs/1704.08863>. Citado na página 63.

LAGARIS, I.; LIKAS, A.; PAPAGEORGIOU, D. Neural-network methods for boundary value problems with irregular boundaries. IEEE transactions on neural networks, v. 11, p. 1041–1049, 2000. Citado 2 vezes nas páginas 25 e 53.

LAGARIS, I. E.; LIKAS, A.; FOTIADIS, D. I. Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks, v. 9, n. 5, p. 987–1000, Sep. 1998. ISSN 1045-9227. Citado 6 vezes nas páginas 24, 49, 52, 53, 60 e 63.

Lau, M. M.; Hann Lim, K. Review of adaptive activation function in deep neural network. In: 2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES). [S.l.: s.n.], 2018. p. 686–690. Citado na página 32.

LEAKE, C.; MORTARI, D. Deep theory of functional connections: A new method for estimating the solutions of partial differential equations. Machine Learning and Knowledge Extraction, MDPI AG, v. 2, n. 1, p. 37–55, Mar 2020. ISSN 2504-4990. Disponível em: <http://dx.doi.org/10.3390/make2010004>. Citado na página 23.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep Learning. [S.l.]: Nature Publishing Group, 2015. 436-444 p. ISBN 978-3-319-94463-0. Citado na página 42.

LEE, H.; KAND, I. S. Neural algorithm for solving differential equations. Journal of Computational Physics, v. 91, n. 1, p. 110–131, 1990. Citado na página 24.

LESHNO, M. et al. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks, v. 6, n. 6, p. 861 – 867, 1993. ISSN 0893-6080. Citado na página 43.

LEVEQUE, R. J. Numerical Methods for Conservation Laws. 2. ed. [S.l.]: Birkhäuser Basel, 1992. Citado na página 78.

LI, X. et al. Integration wavelet neuralnetwork for steady convection dominated diffusion problem. In: 3rd international conference on information and computing. [S.l.: s.n.], 2010. v. 2, n. 2, p. 109–112. Citado 4 vezes nas páginas 16, 84, 85 e 87.

LIU, D. C.; NOCEDAL, J. On the limited memory bfgs method for large scale optimization. MATHEMATICAL PROGRAMMING, v. 45, p. 503–528, 1989. Citado na página 50.

LIU, H. et al. Legendre neural network method for several classes of singularly perturbed differential equations based on mapping and piecewise optimization technology. Neural Processing Letters, v. 51, p. 2891–2913, 2020. Citado na página 86.

LU, Z. et al. The Expressive Power of Neural Networks: A View from the Width. 2017. Citado na página 43.

MARGOSSIAN, C. C. A review of automatic differentiation and its efficient implementation. WIREs Data Mining and Knowledge Discovery, Wiley, v. 9, n. 4, Mar 2019. ISSN 1942-4795. Disponível em: <http://dx.doi.org/10.1002/WIDM.1305>. Citado na página 50.

MCCULLOCH, W.; PITTS, W. A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, v. 5, p. 127–147, 1943. Citado na página 30.

MCFALL, K.; MAHAN, J. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council, v. 20, p. 1221–33, 07 2009. Citado na página 25.

MCFALL, K. S. An Artificial Neural Network Method For Solving Boundary Value Problems With Arbitrary Irregular Boundaries. Tese (Doutorado) — Georgia Institute of Technology, 2006. Citado na página 52.

MCFALL, K. S.; MAHAN, J. R. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. IEEE Transactions on Neural Networks, v. 20, n. 8, p. 1221–1233, 2009. Citado 2 vezes nas páginas 52 e 53.

MEADE, A.; FERNANDEZ, A. The numerical solution of linear ordinary differential equations by feedforward neural networks. Mathematical and Computer Modelling, v. 19, n. 12, p. 1–25, 1994. ISSN 0895-7177. Citado na página 24.

MICHOSKI, C. et al. Solving Differential Equations using Deep Neural Networks. Neurocomputing, 2020. Citado na página 51.

MURTAGH, F. Multilayer perceptrons for classification and regression. Neurocomputing, v. 2, n. 5, p. 183 – 197, 1991. ISSN 0925-2312. Disponível em: <http://www.sciencedirect.com/science/article/pii/0925231291900235>. Citado na página 36.

PARISI, D. R.; LABORDE, M. A. Modeling steady-state heterogeneous gas–solid reactors using feedforward neural networks. Computers Chemical Engineering, v. 25, n. 9, p. 1241 – 1250, 2001. ISSN 0098-1354. Citado na página 25.

PARISI, D. R.; MARIANI, M. C.; LABORDE, M. A. Solving differential equations with unsupervised neural networks. Chemical Engineering and Processing: Process Intensification, v. 42, n. 8, p. 715–721, 2003. ISSN 0255-2701. Citado 3 vezes nas páginas 24, 25 e 49.

PARK, S. et al. Minimum Width for Universal Approximation. 2020. Citado na página 43.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. 12 2019. Citado na página 27.

PINKUS, A. Approximation theory of the mlp model in neural networks. Acta Numerica, Cambridge University Press, v. 8, p. 143–195, 1999. Citado na página 43.

QUARTERONI, A. Numerical Models for Differential Problems. 1. ed. [S.l.]: Springer, Milan, Italy, 2009. Citado 6 vezes nas páginas 23, 46, 69, 76, 78 e 80.

RAISSI, M.; KARNIADAKIS, G. E. Hidden physics models: Machine learning of nonlinear partial differential equations. Journal of Computational Physics, v. 357, p. 125–141, 2018. ISSN 0021-9991. Citado na página 26.

ROLO, R. Aprimoramentos em Modelagem Geológica Implícita com Funções Distância Assinaladas - Proposta de Tese Para o Exame de Qualificação. Tese (Doutorado), 06 2019. Citado 2 vezes nas páginas 15 e 30.

ROOS, H. G.; STYNES, M.; TOBISKA, L. Numerical Methods for Singularly Perturbed Differential Equations. 1. ed. [S.l.]: Springer, Berlin, 1996. Citado 3 vezes nas páginas 23, 78 e 80.

ROSENBLATT, F. The Perceptron, a Perceiving and Recognizing Automaton. [S.l.]: Cornell Aeronautical Laboratory, 1957. (Report: Cornell Aeronautical Laboratory). Citado na página 36.

RUDD, K. Solving Partial Differential Equations Using Artificial Neural Networks. Tese (Doutorado) — Department of Mechanical Engineering and Material Sciences, Duke University, 2013. Citado 5 vezes nas páginas 25, 48, 50, 51 e 52.

SAMANIEGO, E. et al. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. Computer Methods in Applied Mechanics and Engineering, v. 362, p. 112790, 2020. ISSN 0045-7825. Citado na página 26.

SATHYA, R.; ABRAHAM, A. Comparison of supervised and unsupervised learning algorithms for pattern classification. International Journal of Advanced Research in Artificial Intelligence, v. 2, 02 2013. Citado na página 35.

SHIRVANY, Y.; HAYATI, M.; MORADIAN, R. Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations. Applied Soft Computing, v. 9, n. 1, p. 20–29, 2009. Citado na página 25.

SILVA, I. N. d.; SPATTI, D. H.; FLAUZINO, R. A. Redes neurais artificiais para engenharia e ciências aplicadas. [S.l.]: Artliber Editora, 2010. Citado 2 vezes nas páginas 29 e 41.

SILVA, L. H. Monken e; NEITZEL, I.; LIMA, E. P. Resolution of differential equations with artificial neural networks: high gradients and arbitrary domains problems. Acta Scientiarum. Technology, v. 27, n. 1, p. 7–16, 2008. Citado 2 vezes nas páginas 53 e 84.

SIRIGNANO, J.; SPILIOPOULOS, K. Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics, v. 375, p. 1339 – 1364, 2018. ISSN 0021-9991. Citado 4 vezes nas páginas 26, 42, 47 e 51.

SMITH, G. Numerical Solution of Partial Differential Equations: Finite Difference Methods. Clarendon Press, 1985. (Oxford applied mathematics and computing science series). ISBN 9780198596509. Disponível em: <https://books.google.com.br/books?id=hDpvljaHOrMC>. Citado na página 23.

SOYDANER, D. A comparison of optimization algorithms for deep learning. International Journal of Pattern Recognition and Artificial Intelligence, 2020. Citado na página 50.

SUN, S. et al. A survey of optimization methods from a machine learning perspective. IEEE Transactions on Cybernetics, p. 1–14, 2019. Citado na página 50.

TEAM, T. et al. Theano: A python framework for fast computation of mathematical expressions. 05 2016. Citado na página 27.

TINO, P.; BENUSKOVA, L.; SPERDUTI, A. Artificial neural network models. In: _____. Springer Handbook of Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. p. 455–471. ISBN 978-3-662-43505-2. Disponível em: <https://doi.org/10.1007/978-3-662-43505-2_27>. Citado na página 35.

VALLI, A. M. et al. A parameter-free dynamic diffusion method for advection–diffusion–reaction problems. Computers & Mathematics with Applications, v. 75, n. 1, p. 307 – 321, 2018. Citado na página 78.

WANG, P.; FAN, E.; WANG, P. Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. Pattern Recognition Letters, 2020. ISSN 0167-8655. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167865520302981>. Citado na página 42.

WANG, Z. et al. Differential equation analysis on covid-19. Open Access Journal of Clinical Trials, v. 2, p. 1–11, 08 2020. Citado na página 23.

WIDROW, B. Adaptive "Adaline"Neuron Using Chemical "Memistors.". [S.l.]: Stanford University. Stanford Electronics Laboratories. Solid State Electronics Laboratory and United States, 1960. Citado na página 36.

Apêndices

APÊNDICE A – Derivadas da função $N(\cdot,\cdot,\cdot)$

O processo de minimização do funcional resíduo (3.13) é considerado o treinamento da rede neural. O cálculo do resíduo mínimo envolve os valores de saída da rede neural, representados pela função N, e as derivadas desses valores com respeito aos dados de entrada e dos parâmetros de ajuste.

Teorema A.0.1 *Seja $k > 0$ um inteiro positivo e considere a função N descrita por (3.24) e (3.25). Então, as derivadas parciais de N em relação as variáveis (dados de entrada) t e x_s, $s = 1, \cdots, d$ são descritas por*

$$\frac{\partial^k N}{\partial^k t} = \sum_{i=1}^{d_H} v_i w_i^k \phi^{(k)}(z_i), \tag{A.1}$$

$$\frac{\partial^k N}{\partial^k x_s} = \sum_{i=1}^{d_H} v_i w_{is}^k \phi^{(k)}(z_i), \quad s = 1, \cdots, d, \tag{A.2}$$

onde $\phi^{(k)}(z_i) = \frac{\partial^k \phi(z_i)}{\partial {z_i}^k}$.

Demonstração A.1 *Ver (BEIDOKHTI; MALEK, 2009).*

A derivada de qualquer ordem da função N em relação a qualquer dado de entrada pode ser avaliada. O próximo teorema mostra isso.

Teorema A.1.1 *Sejam $\lambda_1, \cdots, \lambda_d$ inteiros positivos e considere a função N descrita por (3.24) e (3.25). Então,*

$$\frac{\partial^{\lambda_0}}{\partial t^{\lambda_0}} \frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}} \frac{\partial^{\lambda_2}}{\partial x_2^{\lambda_2}} \cdots \frac{\partial^{\lambda_d}}{\partial x_d^{\lambda_d}} N = \sum_{i=1}^{d} v_i w_i^{\lambda_0} \Big(\prod_{j=1}^{r} w_{ij}^{\lambda_j} \Big) \phi^{(\alpha)}(z_i), \tag{A.3}$$

onde $\alpha = \sum_{i=0}^{d} \lambda_i$.

Demonstração A.2 *Ver (BEIDOKHTI; MALEK, 2009).*

Observe que as derivadas de qualquer ordem da função N em relação aos dados de entrada são descritas em função das derivadas de mesma ordem da função de ativação ϕ. Dizemos que a derivada da função N foi transferida para a função de ativação ϕ, por isso a função de ativação deve ser suficientemente diferenciável.

Lema A.2.1 *Seja $k > 0$ um inteiro positivo e considere a função de ativação sigmoidal $\phi : \mathbb{R} \longrightarrow \mathbb{R}$,*

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

Essa função possui derivadas de todas as ordens, em relação a x, e sua k-ésima derivada pode ser escrita em termos de $\phi(x)$.

Demonstração A.3 *Ver (BEIDOKHTI; MALEK, 2009).*

Em particular, as 4 primeiras derivadas de $\phi(x)$ são dadas por

$$\phi'(x) = \phi(x)\Big(1-\phi(x)\Big), \tag{A.4}$$

$$\phi''(x) = \phi(x)\Big(2\phi(x)^2-3\phi(x)+1\Big), \tag{A.5}$$

$$\phi'''(x) = \phi(x)\Big(-6\phi(x)^3+12\phi(x)^2-7\phi(x)+1\Big), \tag{A.6}$$

$$\phi^{(4)}(x) = \phi(x)\Big(24\phi(x)^4-60\phi(x)^3+50\phi(x)^2-15\phi(x)+1\Big). \tag{A.7}$$

Vamos determinar também as derivas de N em relação aos parâmetros de ajuste.

Teorema A.3.1 *Se a função N é definida por (3.24) e (3.25), então,*

$$\frac{\partial N}{\partial w_k} = \sum_{i=1}^{d_H} v_i t\phi'(z_i), \tag{A.8}$$

$$\frac{\partial}{\partial w_k}\Big(\frac{\partial N}{\partial t}\Big) = v_k\phi'(z_k)+v_k w_k t\phi'(z_k), \tag{A.9}$$

$$\frac{\partial N}{\partial v_k} = \phi(z_k), \tag{A.10}$$

$$\frac{\partial}{\partial v_k}\Big(\frac{\partial N}{\partial x_j}\Big) = w_{kj}\phi'(z_k) \tag{A.11}$$

APÊNDICE B – Códigos Python

Listagem B.1 – Código implementado em Python com uso da biblioteca Tensorflow para resolução do problema (??)-(??)

```
import tensorflow as tf2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow.compat.v1 as tf


# FUNCIONALIDADE QUE PERMITE A COMPATIBILIDADE COM A VERSÃO 1.5 DO TENSORFLOW
tf.disable_v2_behavior()

#FUNCAO QUE DESCREVE A FUNCAO p(t)
def p(x):
    return x + (1. + 3.*x**2) / (1. + x + x**3)

#FUNCAO QUE DESCREVE A FUNCAO q(t)
def q(x):
    return x**3 + 2.*x + x**2 * ((1. + 3.*x**2) / (1. + x + x**3))

#SOLUCAO EXATA (ESSA SOLUÇÃO ESTÁ ERRADA!!!)
def g_analytic(x):
    return np.exp(-x/5)*np.sin(x)

#QUANTIDADE DE PONTOS (EQUIDISTANTES) DO DOMíNIO
Nx = 10

#RETORNAM VALORES IGUALMENTE ESPAÇADOS DENTRO DO INTERVALO ESPECIFICADO
x = np.linspace(0,1, Nx)

# CONVERTER OBJETO PYTHON PARA OBJETOS DO TIPO TENSOR
x_tf = tf.convert_to_tensor(x.reshape(-1,1),dtype=tf.float64)
g0 = tf.constant(0.2,dtype=tf.float64)
g1 = tf.constant(1.,dtype=tf.float64)
g2 = tf.constant(2.,dtype=tf.float64)
g3 = tf.constant(3.,dtype=tf.float64)
g5 = tf.constant(5.,dtype=tf.float64)

#QUANTIDADE DE ITERAÇÕES DO TREINAMENTO DA REDE
num_iter = 1000

#QUANTIDADE DE NEURÔNIOS EM CADA CAMADA, SÃO DUAS CAMADAS
num_hidden_neurons = [10,10]

#QUANTIDADE DE CAMADAS É DEFINIDO PELA QUANTIDADE DE ELEMENTOS NA LISTA
num_hidden_layers = np.size(num_hidden_neurons)

```

```
45 # CONSTRUÇÃO DA REDE
46 # AGRUPAMENTO DE CADA PASSO NA CONSTRUÇÃO DA REDE NEURAL
47 with tf.name_scope('dnn'):
48
49     # CAMADA DE ENTRADA
50     previous_layer = x_tf
51
52     # CAMADAS ESCONDIDAS
53     for l in range(num_hidden_layers):
54         current_layer = tf.layers.dense(previous_layer, num_hidden_neurons[l], name='
               hidden%d'%(l+1), activation=tf.nn.relu,kernel_initializer='glorot_uniform
               ')
55         previous_layer = current_layer
56
57     # CAMADA DE SAIDA
58     dnn_output = tf.layers.dense(previous_layer, 1, name='output')
59
60 # DEFINIÇÃO DA FUNÇÃO SOLUCAO APROXIMADA E DA SEÇÃO CUSTO
61 with tf.name_scope('custo'):
62     # FUNÇÃO TRIAL
63     g_trial = x_tf * tf.sin(g1) * tf.exp(-g0) + x_tf * (g1 - x_tf) * dnn_output
64
65     # DERIVADA DA FUNÇÃO TRIAL
66     d_g_trial = tf.gradients(g_trial,x_tf)
67
68     right_side = tf.exp(-g_trial/g5)*tf.cos(g_trial) - g_trial - (g1/g5)*d_g_trial[0]
69
70     err = tf.square( d_g_trial[0] - right_side)
71     custo = tf.reduce_sum(err, name = 'custo')
72
73 # ------------------------------------------------------
74 # MÉTODO PARA MINIMIZAR A FUNÇÃO CUSTO
75 # ------------------------------------------------------
76
77 # TAXA DE APRENDIZAGEM
78 learning_rate = 0.001
79 with tf.name_scope('treinamento'):
80     # MÉTODO DE MINIMIZAÇÃO - GRADIENT DESCENT
81     optimizer = tf.train.GradientDescentOptimizer(learning_rate)
82     traning_op = optimizer.minimize(custo)
83
84 g_dnn_tf = None
85
86
87 # DEFINE O NÓ QUE INICIALIZA TODOS OS OUTROS NÓS DO GRAFO - NO CASO, AS CAMADAS
88 init = tf.global_variables_initializer()
89
90 # ETAPA DE CONSTRUÇÃO DO GRAFO - REDE
91 with tf.Session() as sess:
92
93     # INICIALIZA TODA A REDE
94     # EM TODAS AS ETAPAS SERÃO CHAMADAS AS SESSÕES CONSTRUIDAS ACIMA
95     init.run()
```

```
96
97      # INICIO DO TREINAMENTO DA REDE - CHAMA A SESSÃO TREINAMENTO
98      for i in range(num_iter):
99          sess.run(traning_op)
100
101     # APÓS A CONCLUSÃO DO TREINAMENTO TEMOS UMA SOLUÇÃO APROXIMADA
102     # CÁLCULO DO VALOR MINIMO DO FUNCIONAL
103     min_Jr = np.min(err.eval())
104
105     # ARMAZENAMENTO DO RESULTADO FINAL
106     g_dnn_tf = g_trial.eval()
107
108 #SOLUÇÃO ANALITICA
109 g_analytical = g_analytic(x)
110
111 #CÁLCULO DA DIFERENÇA ENTRE SOLUÇÃO POR REDES NEURAIS E SOLUÇÃO ANALITICA
112 diff_tf = g_dnn_tf - g_analytical.reshape(-1,1)
113
114 print("DIFERENÇA ENTRE SOLUÇÃO POR REDES NEURAIS E SOLUÇÃO ANALITICA: ",diff_tf )
```

1. Definer a discretização do intervalo $[a, b]$;

2. Descreva uma função para o cálculo de $f(x, u(x))$;

```
1   # GRADIENTE DA FUNÇÃO DE ATIVAÇÃO
2   def sigmoid_grad(x):
3         return sigmoid(x) * (1 - sigmoid(x))
```

3. Construir a estrutura da rede: camada de entrada;

```
1   # CAMADA DE ENTRADA
2     camada_anterior = pontos_dominio
```

4. Construir a estrutura da rede: camada(s) escondida(s);

```
1   # CAMADAS ESCONDIDAS
2    for l in range(num_camada_oculta):
3        # FUNÇÃO DE ATIVAÇÃO SIGMOIDE, INICIALIZAÇÃO DOS PESOS PELO MÉTODO
             XAVIER
4        camada_atual = tf.layers.dense(camada_anterior,
             num_neuronios_camada_oculta[l],name='oculta%d'%(l+1), activation=tf.
             nn.sigmoid,kernel_initializer='glorot_uniform')
5        camada_anterior = camada_atual
```

5. Construir a estrutura da rede: camada de saída;

```
1   # CAMADA DE SAIDA
2     saida_rede = tf.layers.dense(camada_anterior, 1, name='saida')
```

6. Definir a função aproximada

```
1   # FUNÇÃO APROXIMADA
2   f_aproxi = 1.  + pontos_dominio  * saida_rede
```

7. Definir a derivada da função aproximada

```
# DERIVADA DA FUNÇÃO APROXIMADA
 d_f_aproxi = tf.gradients(f_aproxi,pontos_dominio)
```

8. Calcular o funcional resíduo;

```
q = B(pontos_dominio) - f_aproxi*A(pontos_dominio)
residuo = tf.losses.mean_squared_error(d_f_aproxi[0] - q)
```

9. Definir a taxa de aprendizagem;

```
# TAXA DE APRENDIZAGEM
taxa_aprendizagem = 0.001
```

10. Definir o método de minimização;

```
# MÉTODO DE MINIMIZAÇÃO - DESCIDA DO GRADIENTE
   optimizer = tf.train.GradientDescentOptimizer(taxa_aprendizagem)

   # TREINAMENTO MEDIANTE MINIMIZACAO DO CUSTO
   treinamento = optimizer.minimize(custo)
```

11. Iniciar o treinamento;

```
# INICIO DO TREINAMENTO DA REDE - CHAMA A SESSÃO TREINAMENTO
      for i in range(num_iter):
          sess.run(treinamento)
```

12. Calcular o funcional;

```
# CALCULO DO VALOR MINIMO DO FUNCIONAL
      min_Jr = np.min(erro.eval())
```