UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Aprendizado de máquina aplicado em equações diferenciais com soluções do tipo sóliton com variação de parâmetros.

Igor Andrade Reis de Oliveira

Itajubá, 4 de junho de 2025

UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Igor Andrade Reis de Oliveira

Aprendizado de máquina aplicado em equações diferenciais com soluções do tipo sóliton com variação de parâmetros.

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologias da Computação.

Área de Concentração: Matemática da Computação

Orientador: Prof. Dr. Jeremias Barbosa Machado
Coorientador: Prof. Dr. Alan Bendasoli Pavan

4 de junho de 2025
Itajubá

UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Aprendizado de máquina aplicado em equações diferenciais com soluções do tipo sóliton com variação de parâmetros.

Igor Andrade Reis de Oliveira

Dissertação aprovada por banca examinadora em dia de mês de 2025, conferindo ao autor o título de **Mestre em Ciências em Ciência e Tecnologia da Computação.**

Banca Examinadora:
Profa. Dr. Elisa Gouvea Mauricio Ferreira
Prof. Dr. João Paulo Reus Rodrigues Leite

Itajubá
2025

Igor Andrade Reis de Oliveira
Aprendizado de máquina aplicado em equações diferenciais com soluções do tipo sóliton com variação de parâmetros/ Igor Andrade Reis de Oliveira. – Itajubá, 4 de junho de 2025-
129 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Jeremias Barbosa Machado

Dissertação (Mestrado)
Universidade Federal de Itajubá - UNIFEI
Programa de pós-graduação em ciência e tecnologia da computação, 4 de junho de 2025.

1. Redes Neurais Informadas pela Física. 2. Equações Diferenciais Parciais Parametrizadas. I. Jeremias Barbosa Machado. II. Universidade Federal de Itajubá. III. IESTI. IV. Aprendizado de máquina aplicado em equações diferenciais com soluções do tipo sóliton com variação de parâmetros

CDU 07:181:009.3

Igor Andrade Reis de Oliveira

Aprendizado de máquina aplicado em equações diferenciais com soluções do tipo sóliton com variação de parâmetros

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologias da Computação.

Trabalho aprovado. Itajubá, dia de mês de 2025:

Prof. Dr. Jeremias Barbosa Machado
Orientador

Prof. Dr. Alan Bendasoli Pavan
Coorientador

Profa. Dr. Elisa Gouvea Mauricio Ferreira

Prof. Dr. João Paulo Reus Rodrigues Leite

Itajubá
4 de junho de 2025

Agradecimentos

Agradeço aos meus pais, sem eles não teria a oportunidade de chegar a esta última parte do trabalho, e por isso dedico meus principais agradecimentos.

Agradecimentos especiais ao Dante e à Sarah, meus irmãos, que compartilharam bons memes em todo o processo.

À minha namorada, Mariana, que sempre me apoiou e tornou tudo muito mais leve desde o início do mestrado.

Aos novos amigos que fiz, que tornaram os dias mais agradáveis e fizeram da minha volta a Itajubá uma experiência muito mais divertida, com menções especiais ao Samuel e à Elisa.

Ao meu grupinho de estudos, Daniel, Gabriella e Rogério que, mesmo seguindo caminhos diferentes, a amizade de vocês continuou sendo fundamental.

Ao meu orientador, Jeremias, por apoiar o estudo nesta área que considero tão interessante e por promover reuniões sempre proveitosas, além de sugerir ideias fundamentais que direcionaram o desenvolvimento deste trabalho.

Ao meu co-orientador, Alan, que continuou me acompanhando no estudo mesmo após a orientação anterior, sempre se mostrando disponível e contribuindo para reuniões muito proveitosas.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

"Tudo o que você realmente precisa saber no momento é que o universo é muito mais complicado do que se poderia pensar, mesmo se você começar a partir de uma posição já pensando que ele é muito complicado."

(Douglas Adams)

Resumo

Este trabalho introduz uma versão estendida da estrutura de Redes Neurais Informadas pela Física (PINNs) para resolver equações diferenciais parciais (EDPs) parametrizadas, incorporando uma dimensão adicional de entrada para um parâmetro variável m. Essa melhoria permite que o modelo generalize soluções ao longo de um intervalo específico de parâmetros sem a necessidade de retreinamento para cada valor. Notavelmente, o modelo estendido manteve tempos de treinamento comparáveis ao método padrão ($\approx$280 segundos), demonstrando eficiência computacional apesar da dimensão adicional.

Inicialmente, o modelo foi validado em uma equação diferencial ordinária (EDO) de segunda ordem com coeficientes variáveis, generalizando com eficácia soluções ao longo de intervalos de parâmetros. A investigação prosseguiu para a equação não linear de Korteweg-de Vries (KdV), demonstrando a capacidade do modelo em simular interações complexas de sólitons com dados iniciais limitados. Arquiteturas de rede ótimas foram identificadas para diferentes cenários, enfatizando a importância da escolha dos hiperparâmetros. Embora os vínculos físicos tenham sido diretamente implementados usando o DeepXDE, não foram observadas melhorias significativas na precisão, sugerindo que a equação e as condições iniciais já impõem restrições suficientes.

O foco principal foi a equação de Sine-Gordon, uma EDP não linear com soluções do tipo sóliton, para avaliar a capacidade do modelo em generalizar para o parâmetro variável m. Soluções precisas foram obtidas com dados mínimos, destacando a eficiência do modelo. Desafios surgiram quando $m \approx 0$, onde a equação se aproxima de uma forma de onda linear, reduzindo a capacidade de generalização do modelo. Apesar disso, a abordagem forneceu consistentemente soluções confiáveis na maior parte do intervalo de parâmetros, demonstrando seu potencial para resolver EDPs parametrizadas de forma eficiente.

Palavras-chave: Redes Neurais Informadas pela Física. Equações Diferenciais Parciais Parametrizadas. Equação de Korteweg-de Vries. Equação de Sine-Gordon. Solitons. Aprendizado Profundo. Eficiência Computacional.

Abstract

This work introduces an extended version of the Physics-Informed Neural Networks (PINNs) framework to solve parametrized partial differential equations (PDEs) by incorporating an additional input dimension for a variable parameter m. This enhancement enables the model to generalize solutions across a specified parameter range without retraining for each value. Notably, the extended model maintained training times comparable to the standard approach ($\approx$280 seconds), demonstrating computational efficiency despite the added dimension.

Initially, the model was validated on a second-order ordinary differential equation (ODE) with variable coefficients, effectively generalizing solutions over parameter ranges. The investigation then progressed to the nonlinear Korteweg-de Vries (KdV) equation, demonstrating the model's ability to simulate complex soliton interactions using limited initial data points. Optimal network architectures were identified for different scenarios, emphasizing the importance of hyperparameter selection. Although physical constraints were easily implemented using DeepXDE, no significant accuracy improvements were observed, suggesting that the equation and initial conditions provided sufficient restrictions.

The primary focus was on the Sine-Gordon equation, a nonlinear PDE with soliton solutions, to evaluate the model's ability to generalize for variable parameter m. Accurate solutions were achieved with minimal data, highlighting the model's efficiency. Challenges arose near $m = 0$, where the equation approaches a linear wave form, reducing the model's generalization capacity. Despite this, the approach consistently delivered reliable solutions across most of the parameter range, demonstrating its potential for efficiently solving parametrized PDEs.

Key-words: Physics-Informed Neural Networks. Parametrized Partial Differential Equations. Korteweg-de Vries Equation. Sine-Gordon Equation. Solitons. Deep Learning. Computational Efficiency.

Lista de ilustrações

Figura 1 – A grande área Inteligencia Artificial 24
Figura 2 – Comparação entre programação clássica e aprendizado de máquina . . 25
Figura 3 – Transformação dos dados brutos em uma melhor representação 26
Figura 4 – Transformação de dados por uma rede neural. 27
Figura 5 – *Aprendizado* no aprendizado profundo: a pontuação gerada pela função custo é usada como *feedback* pelo otimizador para atualizar os valores dos pesos. 28
Figura 6 – Fluxograma de uma PINN 30
Figura 7 – Solução equação diferencial de segunda ordem com coeficientes variáveis 57
Figura 8 – Comparação entre os métodos PINN padrão e PINN mais geral. O último incorpora uma dimensão extra de entrada, expandindo o espaço de solução para generalizar sobre uma faixa de valores de parâmetro. . 68
Figura 9 – FNN com duas entradas. 69
Figura 10 – FNN com entrada extra. 69
Figura 11 – Evolução do valor da função custo e métrica L2 durante o treinamento, caso 1: $A \in [0,1], B \in [0,1]$. 75
Figura 12 – Gráfico da evolução dos treinamentos no caso 1: $A \in [0,1], B \in [0,1]$. . 76
Figura 13 – Erro L2 relativo médio para cada par de parâmetros $A \in [0,1], B \in [0,1]$. 77
Figura 14 – Evolução do valor da função custo e métrica L2 durante o treinamento, caso 2: $A \in [-1,1], B \in [-1,1]$. **Obs.**: Para o *log* deste caso específico, uma interação inesperada ocorreu, desconfigurando o mesmo. Como A recuperação deste treinamento em específico seria impossível, foi mantido o gráfico, mesmo que desconfigurado. 78
Figura 15 – Gráfico da evolução dos treinamentos do caso 2: $A \in [-1,1], B \in [-1,1]$. 79
Figura 16 – Erro L2 relativo médio para cada par de parâmetros $A \in [-1,1], B \in [-1,1]$. 80
Figura 17 – Evolução do valor da função custo e métrica L2 durante o treinamento, caso 3: $A \in [0,4], B \in [0,4]$. 81
Figura 18 – Gráfico da evolução dos treinamentos no caso 3: $A \in [0,4], B \in [0,4]$. . 82
Figura 19 – Erro L2 relativo médio para cada par de parâmetros $A \in [0,4], B \in [0,4]$. 83
Figura 20 – Observação sobre o pior caso de avaliação do modelo, no caso 2, com A e B se aproximando de -1. 84
Figura 21 – Evolução do treinamento da rede neural, caso 1 sóliton 86
Figura 22 – Resultado do treinamento de um sóliton 87
Figura 23 – Erro absoluto da solução do tipo 1 sóliton no caso com 4 camadas ocultas de 100 neurônios cada. 88

Figura 24 – Evolução do treinamento da rede neural, caso 2 sólitons 89
Figura 25 – Resultado do treinamento de dois sólitons 90
Figura 26 – Erro absoluto da solução do tipo 2 sólitons no caso com 4 camadas ocultas de 100 neurônios cada. 91
Figura 27 – Gráfico do erro L2 relativo (log_{10}) em relação ao número de camadas e neurônios da rede neural, caso de 1 sóliton. 93
Figura 28 – Gráfico do erro L2 relativo (log_{10}) em relação ao número de camadas e neurônios da rede neural, caso de 2 sólitons. 94
Figura 29 – Modelo com condição inicial relativa a dois sólitons, com uma camada oculta de 120 neurônios. 94
Figura 30 – Modelo com condição inicial relativa a dois sólitons, com duas camadas ocultas de 20 neurônios. 95
Figura 31 – Gráfico do erro L2 relativo (log_{10}) em relação ao número de camadas e neurônios da rede neural, caso de 1 sóliton em que o vínculo extra relativo a segunda lei de conservação foi utilizado. 96
Figura 32 – Gráfico do erro L2 relativo (log_{10}) em relação ao número de camadas e neurônios da rede neural, caso de 2 sólitons em que o vínculo extra relativo a segunda lei de conservação foi utilizado. 99
Figura 33 – Evolução do treinamento da rede neural, caso 1 sóliton em que foi realizado um treinamento mais robusto. 99
Figura 34 – Resultado do treinamento mais robusto de um sóliton 101
Figura 35 – Erro absoluto da solução do tipo 1 sóliton no caso com 8 camadas ocultas de 20 neurônios cada. 102
Figura 36 – Evolução do treinamento da rede neural, caso 2 sólitons 102
Figura 37 – Resultado do treinamento de dois sólitons pelo modelo mais robusto e otimizado. 103
Figura 38 – Erro absoluto da solução do tipo 2 sólitons no caso com 2 camadas ocultas de 40 neurônios cada. 104
Figura 39 – Colisão de dois sólitons: Na colisão de dois sólitons, observa-se que, antes da colisão, o sóliton de maior amplitude, devido à sua maior velocidade, aproxima-se e colide com o sóliton de menor amplitude. Após a colisão, ambos os sólitons continuam seu movimento sem alteração em sua forma ou tamanho. 105
Figura 40 – Resultados do modelo da equação de Sine-Gordon com o parâmetro m fixado em 0,3 treinado por 20 mil épocas, com refinamento posterior via otimização L-BFGS. 107
Figura 41 – Evolução do erro L2 relativo em função do raio de intervalo em torno de m=0,3. 108

Figura 42 – Resultados do modelo treinado, fixando $m = 0{,}05$, no domínio espaço temporal x e t. 109
Figura 43 – Resultados do modelo treinado, fixando $m = 0{,}175$, no domínio espaço temporal x e t. 110
Figura 44 – Resultados do modelo treinado, fixando $m = 0{,}3$, no domínio espaço temporal x e t. 110
Figura 45 – Resultados do modelo treinado, fixando $m = 0{,}425$, no domínio espaço temporal x e t. 111
Figura 46 – Resultados do modelo treinado, fixando $m = 0{,}55$, no domínio espaço temporal x e t. 111
Figura 47 – Resultado do erro relativo L2 em relação ao parâmetro m, no domínio espaço temporal $\{(x,t) \mid x \in [-5,5], t \in [0,2]\}$ 112
Figura 48 – Resultado do modelo da equação de Sine-Gordon com o parâmetro m fixado em 0,175, treinado por 200 mil épocas para o intervalo $m \in [0{,}05, 0{,}55]$, e seus respectivos erros. 113
Figura 49 – Resultado do modelo da equação de Sine-Gordon com o parâmetro m fixado em 0,425, treinado por 200 mil épocas para o intervalo $m \in [0{,}05, 0{,}55]$, e seus respectivos erros. 113
Figura 50 – Comparação do erro L2 relativo, nos casos de 3 e 4 condições de contorno (3 BC e 4 BC, respectivamente), calculados no domínio espaço temporal $\{(x,t) \mid x \in [-5,5], t \in [0,2]\}$, em relação ao parâmetro fixo m. 114
Figura 51 – Resultados do modelo da equação de Sine-Gordon com o parâmetro m fixado em 0,5 treinado por 20 mil épocas. 115
Figura 52 – Evolução do erro L2 relativo em função do raio de intervalo em torno de $m = 0{,}5$. Caso do modelo que conta com 4 condições de contorno para a equação de Sine-Gordon com parâmetro m variável. 116
Figura 53 – Resultados do modelo treinado com $m = 0.04$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida. 117
Figura 54 – Resultados do modelo treinado com $m = 0.27$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida. 117
Figura 55 – Resultados do modelo treinado com $m = 0.50$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida. 118
Figura 56 – Resultados do modelo treinado com $m = 0.73$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida. 118
Figura 57 – Resultados do modelo treinado com $m = 0.96$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida. 118
Figura 58 – Resultado do erro relativo L2 em relação ao parâmetro m, no domínio espaço temporal $\{(x,t) \mid x \in [-5,5], t \in [0,2]\}$, para o modelo com 4 condições de contorno. 119

Lista de tabelas

Tabela 1 – Comparação de Aspectos Metodológicos 73
Tabela 2 – Resultados dos testes - Parte 1 92
Tabela 3 – Resultados dos testes - Parte 2 93
Tabela 4 – Resultados dos testes com o uso de leis de conservação - Parte 1 97
Tabela 5 – Resultados dos testes com o uso de leis de conservação - Parte 2 98

Sumário

1 INTRODUÇÃO 19
1.1 Organização do trabalho 21

2 REVISÃO TEÓRICA 22
2.1 Inteligência Artificial, Aprendizado de Máquina, Aprendizado Profundo 22
2.1.1 Uma breve história da Inteligência artificial 22
2.1.2 Aprendizado de Máquina 24
2.1.3 Aprendizado Profundo 26
2.2 Redes neurais informadas pela física 29
2.2.1 Solução de equações diferencias parciais não lineares com redes neurais informadas pela física 30
2.2.2 Implementação 33
2.3 DeepXDE 34
2.3.1 Definição do Domínio Computacional 34
2.3.2 Definição da Equação Diferencial Parcial 35
2.3.3 Condições de Contorno e Iniciais 35
2.3.4 Combinação dos Componentes do Problema 36
2.3.5 Construção da Rede Neural 36
2.3.6 Definição e Compilação do Modelo 37
2.3.7 Treinamento e Realização de Predições 37
2.3.8 Termos em Treinamento de Redes Neurais 38
2.4 Equações Diferenciais 38
2.4.1 Equação Diferencial de Segunda Ordem 39
2.4.1.1 Aplicações 40
2.4.1.2 Solução Geral 40
2.4.1.3 Solução Hiperbólica 41
2.4.2 Equações Diferenciais Parciais 42
2.4.3 Determinação do Número de Condições 42
2.4.4 Problemas bem-postos (Well-Posedness) 43
2.4.4.1 Problemas bem-postos e PINNs 44
2.5 Sólitons 45
2.5.1 Equação de Korteweg-De Vries 47
2.5.1.1 Leis de Conservação da Equação KdV 49
2.5.2 Equação Sine-Gordon 49

3 METODOLOGIA . . . 52
3.1 Equação diferencial ordinária de segunda ordem com coeficientes variáveis . . . 52
3.1.1 Elucidando o problema . . . 52
3.1.2 Implementação . . . 53
3.1.3 Validação dos resultados . . . 57
3.2 Soluções do tipo sóliton da Equação KdV . . . 58
3.2.1 Elucidando o problema . . . 59
3.2.2 Implementação . . . 59
3.2.2.1 Implementação do caso original da equação KdV para um e dois sólitons . . . 59
3.2.3 Otimização da arquitetura da rede neural . . . 63
3.2.3.1 Implementação . . . 63
3.2.4 Leis de conservação . . . 64
3.2.4.1 Implementação . . . 65
3.3 Generalização da Equação de Sine-Gordon . . . 66
3.3.1 Arquitetura da rede neural utilizada . . . 68
3.3.2 Função resíduo, condições iniciais e de fronteira . . . 70
3.4 Métricas de Avaliação . . . 71
3.5 Análise Comparativa de Metodologias para EDPs Parametrizadas . . . 71
3.5.1 Objetivo Principal e Tratamento de Parâmetros . . . 71
3.5.2 Arquitetura da Rede, Treinamento e Predição . . . 72
3.5.3 Simplicidade Conceitual e Esforço de Implementação . . . 72
3.5.4 Resumo dos Pontos Fortes . . . 72

4 RESULTADOS E DISCUSSÕES . . . 74
4.1 Equação diferencial ordinária de segunda ordem com coeficientes variáveis . . . 74
4.1.1 Caso 1 . . . 75
4.1.2 Caso 2 . . . 78
4.1.3 Caso 3 . . . 80
4.1.4 Discussões . . . 83
4.2 Solitons da equação KdV . . . 85
4.2.1 Implementação direta do problema . . . 85
4.2.1.1 Solução de um sóliton da equação KdV via redes neurais . . . 85
4.2.1.2 Solução de dois sólitons da equação KdV via redes neurais . . . 88
4.2.2 Testes sistemáticos da equação KdV . . . 91
4.2.2.1 Arquitetura mínima para a distinção de dois sólitons pela máquina . . . 94
4.2.2.2 Caso de vinculo extra relativo a leis de conservação . . . 96
4.2.3 Treinamento robusto a partir da arquitetura de melhor resultado . . . 98
4.2.3.1 Equação KdV com um sóliton . . . 98

4.2.3.2 Equação KdV com dois sólitons 100
4.3 Sine-Gordon 106
4.3.1 Sine-Gordon utilizando 3 condições de contorno 106
4.3.1.1 Resultados para variações do intervalo do parâmetro m 107
4.3.1.2 Treinamento robusto a partir dos testes sistemáticos 108
4.3.2 Resultados utilizando uma condição de contorno extra 114
4.3.2.1 Comparação com o resultado com três condições de contorno 114
4.3.2.2 Avaliação do modelo com 4 condições de contorno em intervalos ampliados do parâmetro m 115
4.3.2.3 Treinamento robusto do modelo com 4 condições de contorno 116
4.3.3 Análise do Tempo de Treinamento e Impacto da Dimensão Extra de Entrada 120

5 CONCLUSÃO 121

REFERÊNCIAS 124

1 Introdução

O uso de redes neurais como uma ferramenta para solução de equações diferencias através das *Physics-informed neural networks* (PINNs) [1] tem se mostrado uma área de grande atividade na atualidade, sendo observado um rápido aumento no número de artigos publicados, sendo aproximadamente 30 em 2018, 100 em 2019, 600 em 2020 e 1300 em 2021 [2]. O método se destaca por aplicar vínculos físicos nas redes neurais, limitando as generalizações possíveis do modelo. Com o uso de conhecimentos prévios da física, essas redes podem representar soluções de equações diferenciais ou até mesmo buscar essas equações. Ambas as tarefas podem ser realizadas a partir de uma quantidade relativamente pequena de dados. Fazendo uso do método, é possível explorar uma forma alternativa de se lidar com equações diferenciais, entre elas, equações diferenciais parciais não lineares.

Equações diferenciais parciais (EDP) não lineares apresentam desafios na sua busca de soluções, sendo que, no geral, os métodos de soluções utilizados em equações lineares não se aplicam. O princípio de superposição de soluções deixa de ser válido, de modo que qualquer método que faça uso deste princípio passa a não ser utilizável no caso de uma equação não linear [3]. Desta forma, métodos alternativos passam a ser uma adição valiosa ao se estudar equações com estas características, sendo a PINN um método promissor a ser estudado.

Sólitons representam um tipo de solução presente em alguns tipos de EDP não lineares. Eles são ondas não lineares que possuem características únicas, mantendo sua forma e velocidade mesmo ao se propagarem em meios onde efeitos não lineares e dispersivos estão presentes. Sólitons possuem uma vasta aplicabilidade, em várias áreas diferentes, sendo usados em sistemas de comunicação para transmissão de sinais a longas distâncias em fibras óticas ou na biologia como uma descrição da propagação de sinais em biomembranas no sistema nervoso. Também são utilizados na dinâmica dos fluidos para se entender o comportamento de ondas, ainda possuindo aplicações na mecânica quântica, física de plasma, astrofísica, biologia molecular, tornando sua versatilidade uma ferramenta valiosa para aplicação em uma ampla gama áreas de estudo [4].

Em uma revisão bibliográfica sobre o tema foram analisadas, no primeiro semestre de 2023, aplicações de redes neurais no contexto de EDPs não lineares que admitem soluções do tipo sóliton, observando mais especificamente o caso de uso nas equações KdV [5] e Sine-Gordon [6]. Foram encontrados estudos referentes à equação KdV nos trabalhos [7, 8, 9, 10, 11, 12, 13, 14, 15, 16] e de forma relacionada nos trabalhos [17, 18]. E em relação à equação de Sine-Gordon nos trabalhos [19, 20], sendo o último um estudo

envolvendo também a equação KdV, em que foi notada uma escassez de estudos com a equação Sine-Gordon.

Os artigos abordam diversas formas de se utilizar redes neurais no contexto de sólitons, incluindo modificações, aprimoramentos, ou até mesmo partindo de outras abordagens diferentes do método das PINNs. Algo que foi notado é a forma de se considerar sempre os parâmetros da equação diferencial como fixos, quando uma solução específica da equação diferencial é buscada.

Por outro lado, observou-se que variações nos parâmetros da equação diferencial não foram abordadas. Com o intuito de se testar a capacidade de generalização das redes neurais para soluções de equações diferenciais, num contexto em que parâmetros podem ser variados, foi proposta uma abordagem diferente. Ao invés de se utilizar o método para a resolução da equação para um parâmetro fixo, dado um domínio espaço temporal, foi utilizada a capacidade de generalização das redes neurais para se resolver uma EDP para um intervalo de parâmetros dado um domínio espaço temporal.

Objetivos

Este trabalho tem como objetivo resolver equações diferenciais em um intervalo de parâmetros, de modo que não se obtenha apenas uma solução específica, mas sim uma família de soluções correspondente a diferentes configurações do problema. Assim, ao final do treinamento, o modelo seria capaz de generalizar para diferentes valores de parâmetros sem a necessidade de recomeçar o processo de aprendizagem.

Etapas

Inicialmente, foi proposto um experimento com uma equação diferencial ordinária (EDO) de segunda ordem, permitindo a variação de dois de seus coeficientes dentro de intervalos predefinidos. Este estudo preliminar teve como objetivo avaliar a capacidade do modelo em lidar com parâmetros variáveis em um contexto mais simples.

Em seguida, foi conduzido um estudo de caso com uma equação diferencial parcial (EDP) não linear que admite soluções do tipo sóliton: a equação de Korteweg-de Vries (KdV). Nessa etapa, utilizou-se a biblioteca DeepXDE, escrita em *Python*, com o intuito de familiarizar-se com a modelagem de EDPs não lineares. Embora os coeficientes da equação tenham sido mantidos fixos, investigou-se o impacto da variação de hiperparâmetros da rede e de vínculos físicos adicionais [11].

Com base nas abordagens anteriores, partiu-se para o estudo principal: a aplicação de redes neurais informadas pela física (PINNs) a uma equação não linear com coeficientes variáveis e soluções do tipo sóliton. Escolheu-se a equação de Sine-Gordon, em razão da escassez de estudos utilizando PINNs para esse modelo. O objetivo final foi avaliar a

capacidade de generalização do modelo para diferentes valores de coeficientes e compará-la ao desempenho obtido no caso tradicional de parâmetros fixos.

1.1 Organização do trabalho

O capítulo 2 deste trabalho é dedicado à revisão da teórica necessária para uma compreensão do trabalho, sendo abordados inicialmente tópicos de Inteligência Artificial, abrangendo também Aprendizado de Máquina e Aprendizado Profundo. Seguido por uma apresentação das chamadas Redes neurais informadas pela física (PINNs), método utilizado no trabalho para a solução das equações diferenciais. E por último, apresenta-se também uma revisão teórica da física de sólitons e das equações KdV e Sine-Gordon.

Em seguida, no capítulo 3, é abordada a metodologia utilizada no desenvolvimento do trabalho, sendo iniciado pelo caso de implementação de uma rede neural informada pela física no caso de uma equação diferencial ordinária, porém, com coeficientes variáveis. Em seguida, aborda-se o caso da equação KdV com coeficientes fixos, porém abordando casos de variação de hiper parâmetros e a adição de vínculos físicos extras. A última parte do capítulo se trata da metodologia utilizada no caso da solução da equação de Sine-Gordon, contudo, na presença de coeficientes variáveis.

No capítulo 4 são apresentados os resultados obtidos a partir do capítulo anterior e também as discussões referentes a eles. O trabalho é finalizado no capítulo 5, com a apresentação das conclusões.

2 Revisão teórica

Nesse capítulo são apresentados os tópicos teóricos necessários para uma boa compreensão do desenvolvimento deste trabalho. Inicia-se com uma breve revisão sobre redes neurais artificiais, seguida pela apresentação das *PINN*s e do *framework DeepXDE* utilizados para resolver as equações diferenciais propostas.

2.1 Inteligência Artificial, Aprendizado de Máquina, Aprendizado Profundo

A Inteligência Artificial (IA) já faz parte do imaginário popular, e segundo Chollet, já foi sujeita a grande campanha publicitária da mídia [21]. Aprendizado de máquina, aprendizado profundo e IA aparecem de diversas formas, seja na literatura especializada, em divulgação científica ou em obras de ficção científica. Promessas exorbitantes muitas vezes são feitas, em que utopias são idealizadas ou, outras vezes, distopias geradas pela IA são temidas. É importante, ao se estudar e executar ações na área, estar atento a esses detalhes e conseguir distinguir afirmações exageradas de descobertas importantes que podem realmente mudar o mundo como conhecemos. Faz-se necessário conhecer um pouco da história da inteligência artificial para se entender um pouco mais da importância de se saber fazer essas distinções.

2.1.1 Uma breve história da Inteligência artificial

O conceito de Inteligência Artificial nasceu em meados de 1950, quando os pioneiros da área emergente de ciência da computação começaram a se questionar sobre a possibilidade de se fazer os computadores "pensarem". Esta questão e suas ramificações são exploradas até o hoje.

De acordo com Benko e Lányi [22], a inteligência artificial (IA) foi cristalizada como área de pesquisa em 1956, quando John McCarthy definiu o termo. Foi a primeira vez que o tema capturou a atenção de pesquisadores e foi discutido numa conferência em Dartmouth. No ano seguinte, o primeiro solucionador geral de problemas foi testado, e pouco mais de um ano depois, McCarthy, considerado o pai da IA, anunciou a linguagem Lisp, usada para criar programas de IA. Lisp, que tem como significado o processamento de listas, é usada regularmente até os dias atuais.

Em 1965 Herbert Simon disse: "Máquinas serão capazes, daqui a 20 anos, de fazer qualquer trabalho que um homem consiga fazer" [21]. Porém, alguns anos depois, cientistas chegaram à conclusão de que um algoritmo capaz de fazer qualquer coisa que um humano

consiga é praticamente impossível. Atualmente, IA tem um novo significado: criar agentes inteligentes para executar nossos trabalhos de forma mais rápida e fácil.

Em 1968, Marvin Minsky e Seymour Papert demonstraram os limites de uma rede neural simples, publicando que computadores, na época, não possuíam poder de processamento suficiente para lidar com a complexidade de uma rede neural artificial [23]. Em 1970, ocorreu a primeira Conferência Internacional Conjunta sobre Inteligência Artificial (IJCAI), realizada em Washington, DC.

A história da IA está cheia de altos e baixos, onde cortes significativos em investimentos ocorreram, como por exemplo em 1973, quando o congresso dos Estados Unidos criticou os altos gastos com a pesquisa no campo de IA . Seguido no mesmo ano, o matemático inglês James Lighthill publicou um relatório questionando o olhar otimista dado para a área pelos pesquisadores. Segundo Lighthill, máquinas só seriam capazes de alcançar níveis de "amadores experientes" em jogos como xadrez, e que raciocínio de senso comum seria para sempre algo inalcançável. Como resposta, o governo britânico cortou investimentos nesta área nas universidades, exceto nas universidades de Edinburgh, Sussex e Essex. O governo estadunidense rapidamente seguiu o exemplo. Os cortes nos investimentos perduraram até 1980, quando, como resposta a investimentos pesados do Japão em IA, investimentos foram retomados nos Estados Unidos [24].

Em 1997, o programa da IBM, chamado Deep Blue, capaz de prever a condição ideal de movimento de um jogo de xadrez, olhando 20 movimentos à frente e capaz de processar 200 milhões de possibilidades por segundo, venceu o campeão mundial Gary Kasparov, provando assim errada a declaração de Lighthill. A relativa falta de desenvolvimento se deu em parte por causa de como os primeiros sistemas operavam. Em sua maior parte, eram os chamados "sistemas especialistas" conceito que assume que a inteligência humana pode ser formalizada com um conjunto de regras "se-então" e, para áreas em que é possível tal feito, como por exemplo, xadrez, as máquinas se demonstravam bastante úteis.

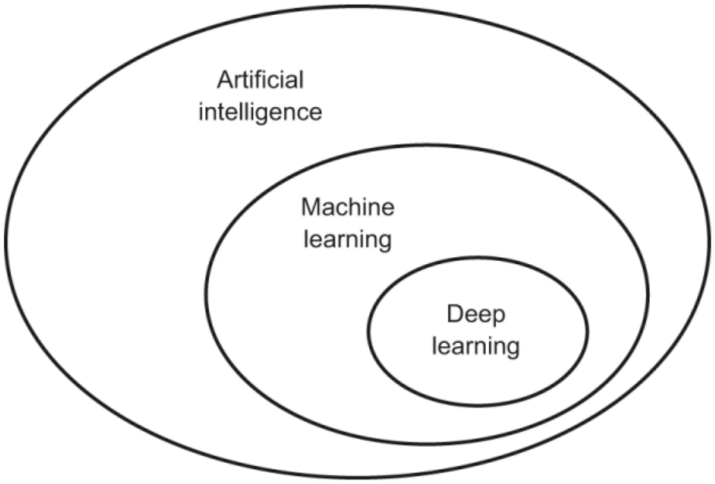
Em áreas que tal feito não é possível, porém, como por exemplo reconhecimento de imagens, tal feito de se simplificar em regras simples não era observado. Para tais tarefas, é necessário que o sistema interprete dados externos corretamente, aprenda com tais dados e use este aprendizado para conquistar objetivos definidos através de uma adaptação flexível. Tais feitos não eram observados pelos chamados sistemas especialistas que, desta forma, poderiam ser considerados como uma forma de pseudo inteligência artificial.

As redes neurais artificiais tiveram uma volta ao centro de atenções por sua vez em 2015, na forma de *Deep Learning* (aprendizado profundo), quando o AlphaGo, um programa desenvolvido pelo Google, conseguiu derrotar o campeão mundial do jogo de tabuleiro Go, jogo substancialmente mais complexo do que xadrez, uma vez que, ao invés de 20 jogadas iniciais possíveis, Go conta com 361 jogadas.

2.1.2 Aprendizado de Máquina

Segundo [21] de forma concisa, IA pode ser descrita como o esforço de automatizar tarefas intelectuais normalmente feitas por humanos. Desta forma, a IA é uma área mais geral que abriga outras áreas como Aprendizado de Máquina e Aprendizado Profundo (Figura 1), porém também inclui várias outras áreas que podem não envolver nenhuma forma de "aprendizado" . Até em meados de 1980 a palavra "aprendizado" não aparecia nos livros de IA de forma nenhuma. Por muito tempo, a IA consistia em regras criadas por programadores e não consistia de forma alguma de um aprendizado de máquina, mesmo que para tarefas em que isso é possível de ser feito de forma eficaz, como nas máquinas especialistas citadas anteriormente.

Figura 1 – A grande área Inteligencia Artificial (*Artificial Intelligence*)



Fonte: Chollet [21].

Para áreas sem regras bem definidas, como classificação de imagens, reconhecimento de voz ou tradução de linguagem natural, uma abordagem diferente se faz necessária.

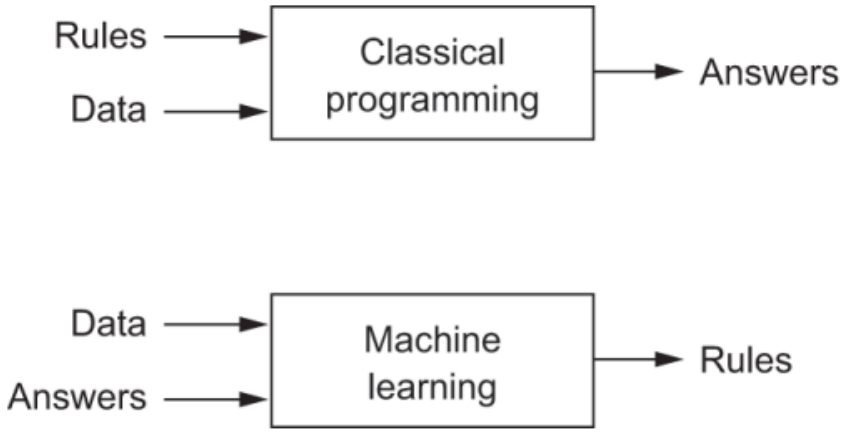
Um sistema de aprendizado de máquina é treinado ao invés de explicitamente programado. São apresentados vários exemplos relevantes para uma tarefa, e é esperado que o sistema consiga achar estruturas estatísticas nesses exemplos de forma que eventualmente o sistema consiga apresentar regras para automatizar uma tarefa, como mostrado na Figura 2.

Para tanto, faz-se necessário definir como máquinas conseguem aprender regras e representações a partir de dados. Para se fazer o aprendizado de máquina, são necessárias 3 coisas:

- Pontos de Entrada de Dados - Por exemplo, imagens para uma tarefa de classificação de imagens

- Exemplos de saída esperados - No exemplo de imagens, por exemplo “gato”, “cachorro”, etc
- Uma regra para se medir se o algoritmo está fazendo um bom trabalho. Esta etapa é necessária para se determinar a distância entre a saída atual e o valor esperado. Isto é, um *feedback* para se ajustar o funcionamento do algoritmo. Tal etapa de ajuste cíclico é o que é chamado de “aprendizado”.

Figura 2 – Comparação entre programação clássica (*Classical programing*) e aprendizado de máquina (*Machine learning*)



Fonte: Chollet [21].

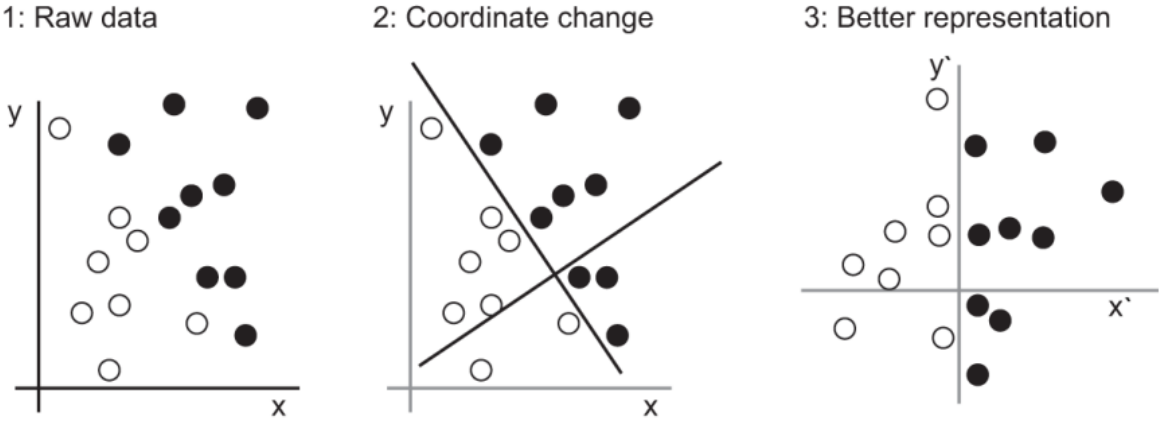
Um modelo de aprendizado de máquina transforma dados de entrada em saídas significativas. Essa transformação de dados é um aspecto central do processo, pois permite extrair padrões da organização dos dados. Além disso, é fundamental que o modelo aprenda representações úteis dos dados de entrada disponíveis.

Num exemplo simples de classificar os pontos entre brancos e pretos, em que como entrada é dada uma coordenada (x,y) e como saída é dado se é mais provável ser um ponto branco ou preto. É notável que, a partir de uma mudança de coordenada representada na Figura 3 esta tarefa pode se tornar bem mais simples.

Com uma melhor representação dos dados para a tarefa em questão, é possível classificar os pontos em questão com uma simples regra: “caso $x' > 0$ o ponto em questão é um ponto preto, ou caso $x' < 0$ o ponto é branco”. Tal melhor representação foi feita a partir de inteligência humana. E se fosse tentada uma forma de automatização? Caso uma procura sistemática de diferentes tipos de representações fosse feita sobre os dados em questão, usando como *feedback* a porcentagem de pontos descritos corretamente como pretos ou brancos? Tal feito seria considerado uma tarefa de aprendizado de máquina.

Um algoritmo de aprendizado de máquina não é usualmente criativo ao se procurar estas transformações, ele estaria apenas procurando entre um conjunto predefinido de

Figura 3 – Transformação dos dados brutos (*raw data*) em uma melhor representação (*better representation*) por meio de uma mudança de coordenadas (*coordinate change*).



Fonte: Chollet [21].

operações aquela que melhor transforma os dados de forma a melhor representá-los para a tarefa requisitada. Este conjunto de operações será chamado de "espaço de hipóteses".

Em suma, aprendizado de máquina é a busca por representações e regras úteis para um conjunto de dados de entrada, dentro de um espaço predefinido de possibilidades, guiado pelo sinal de um feedback. Vale destacar que existem outras abordagens, como o aprendizado não supervisionado, que não dependem de feedback, mas o foco deste trabalho está no aprendizado supervisionado. Esta ideia simples pode ser usada para se resolver uma gama notavelmente ampla de tarefas intelectuais, do reconhecimento de voz à direção autônoma [21].

Uma vez que foi definido o conceito de aprendizado, podemos explorar a subárea de aprendizado de máquina, representado na Figura 1, chamada de aprendizado profundo (*deep learning*).

2.1.3 Aprendizado Profundo

A palavra profundo em aprendizado profundo não está ligada a um maior aprendizado alcançado pelo método, mas sim na ideia de camadas sucessivas de representações. É esperado neste método que o aprendizado na representação dos dados se dê a partir de sucessivas camadas de representação cada vez mais significativas. Desta forma, o número de camadas que contribuem para o modelo é chamado de profundidade do modelo. Modelos de aprendizado profundo são compostos de dezenas ou até mesmo centenas de camadas sucessivas de representação, em que todas passam pelo processo de "aprender" automaticamente a partir dos dados expostos. Existem também abordagens com menos camadas, as quais são chamadas de aprendizado raso (*shalow learning*).

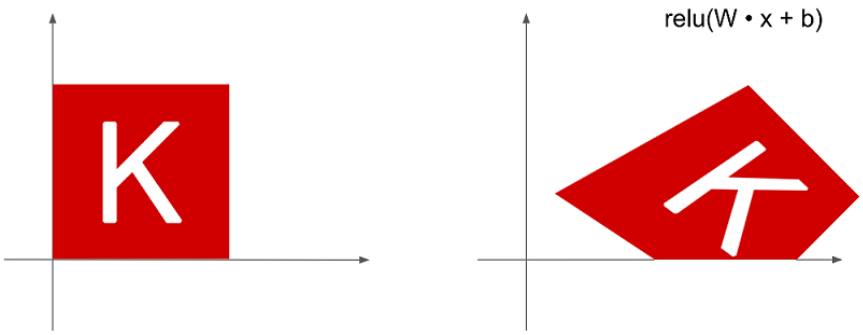
Tais camadas de representação no aprendizado profundo passam pelo processo de

aprendizado utilizando modelos chamados de redes neurais (*neural networks*). O termo rede neural é uma referência à neurobiologia, com uma inspiração no entendimento do cérebro. Porém, é importante esclarecer que modelos de aprendizado profundo não são modelos de um cérebro; não há evidência de que o cérebro use os mecanismos usados pelos modelos modernos de aprendizado profundo.

Como comentado anteriormente, aprendizado de máquina consiste no mapeamento entre entradas e saídas, realizado com base na observação de múltiplos exemplos rotulados. Redes neurais profundas implementam esse mapeamento por meio de uma sequência de transformações sucessivas e simples sobre os dados (camadas). Essas transformações são ajustadas com base na exposição a exemplos.

O aprendizado ocorre por meio da aplicação de operações lineares — geralmente multiplicações de matrizes (ou, de forma mais geral, tensores) entre as entradas e os pesos (*weights*) das camadas — seguidas de somas com termos de *bias* e da aplicação de funções de ativação não lineares. Esse processo é ilustrado na Figura 4.

Figura 4 – Transformação de dados por uma rede neural.



Fonte: Chollet [21].

Ao se representar graficamente um modelo bidimensional, observa-se que as primeiras operações realizadas são bem usuais: uma multiplicação de matrizes pode rotacionar, esticar ou comprimir os dados, enquanto a adição do termo *bias* corresponde a uma translação. A função de ativação, no entanto, introduz uma operação não linear – como no caso da ReLU, que transforma os valores negativos em zero – que não seria possível com transformações puramente lineares.

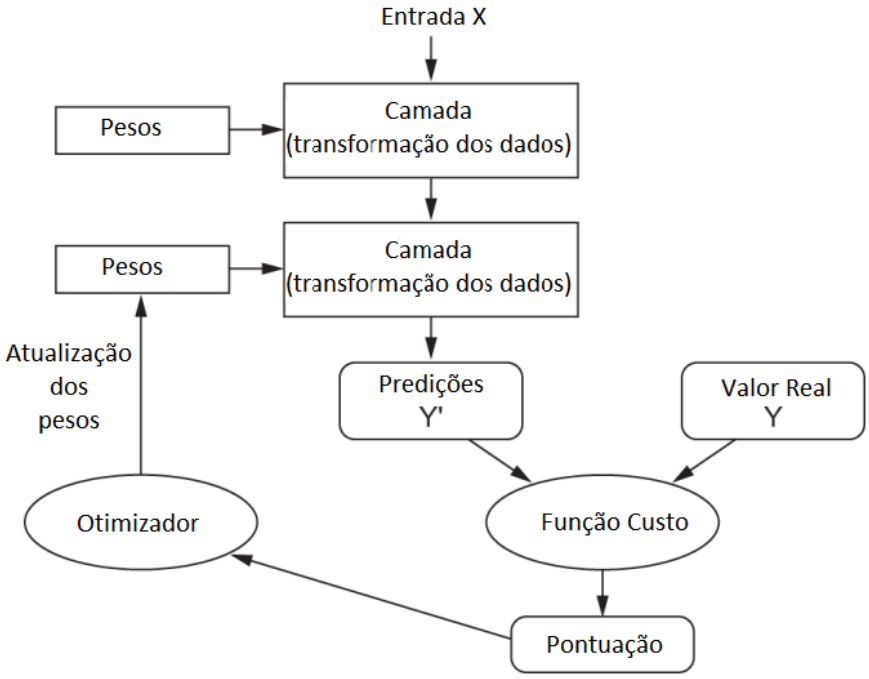
Como descrito em [21], o objetivo de uma rede neural é generalizar da melhor forma possível, realizando transformações significativas nos dados de entrada. Ao restringir o modelo a transformações lineares, impõe-se um espaço de hipóteses limitado. Além disso, é possível demonstrar que uma única transformação linear pode ser equivalente a uma sequência de múltiplas transformações lineares. Dessa forma, o uso de várias camadas lineares não contribui, de fato, para o aumento da complexidade do modelo.

Para lidar com esta limitação de transformações lineares, são adicionadas as funções de ativação, que atuam nas operações anteriores, aumentando a complexidade do

modelo ao se adicionar novas camadas e expandindo o espaço de hipóteses do modelo para o caso não linear.

Uma vez que as transformações em cada camada foram explicadas, um esquema representando um ciclo de aprendizado de uma rede neural foi representado na figura 5.

Figura 5 – *Aprendizado* no aprendizado profundo: a pontuação gerada pela função custo é usada como *feedback* pelo otimizador para atualizar os valores dos pesos.



Fonte: Chollet (2021), editado pelo autor.

As transformações realizadas por uma camada sobre suas entradas são determinadas por um conjunto de parâmetros treináveis conhecidos como pesos (*weights*). Esses pesos representam os coeficientes que definem as operações lineares aplicadas aos dados de entrada. Em outras palavras, cada camada é parametrizada por seus próprios pesos, os quais controlam como as informações são processadas ao longo da rede.

Dessa forma, o processo de aprendizado consiste em ajustar esses pesos de maneira a minimizar o erro entre as saídas produzidas pela rede e os alvos esperados. No entanto, um desafio importante surge: redes neurais modernas podem conter dezenas de milhões de parâmetros. Ajustar tais valores de forma eficiente é uma tarefa complexa, visto que a modificação de um único peso pode influenciar o comportamento geral da rede [21].

Para que seja possível ajustar os parâmetros de uma rede neural, é necessário dispor de uma métrica que quantifique o quão distante está a predição gerada em relação ao valor esperado. Essa função de avaliação, denominada função de custo (*loss function*), atribui uma pontuação que representa essa discrepância, permitindo mensurar o desempenho da rede em determinada tarefa. A partir desse valor escalar, os algoritmos de otimização podem orientar o processo de aprendizado, ajustando os pesos de modo a minimizar o

erro cometido.

Uma vez definida uma forma de avaliar o desempenho do aprendizado profundo, utiliza-se esse sinal de *feedback* para ajustar os pesos na direção que reduza o valor da função custo. Como essa pontuação representa a distância entre o valor previsto e o valor esperado, os ajustes aproximam a saída da rede ao resultado desejado. Tal atualização é realizada pelo otimizador (*optimizer*), que implementa o algoritmo de retropropagação (*backpropagation*), o principal mecanismo de aprendizado em redes neurais profundas. A retropropagação calcula o gradiente da função custo em relação aos pesos, identificando a direção de maior redução. A taxa de aprendizado (*learning rate*) controla o tamanho do passo dado nessa direção, influenciando diretamente a eficiência e a estabilidade da otimização.

Inicialmente, os pesos da rede são atribuídos de forma aleatória, o que implica que, neste estágio, a rede realiza apenas uma sequência de transformações arbitrárias sobre os dados de entrada. Como consequência, as saídas produzidas tendem a estar significativamente distantes dos valores esperados, resultando em uma pontuação elevada da função de custo. A partir dessa avaliação, o algoritmo de otimização ajusta os pesos na direção que reduz o erro, promovendo pequenas melhorias no desempenho da rede.

Esse processo iterativo constitui o chamado ciclo de treinamento. À medida que ele é repetido, espera-se que a função de custo seja progressivamente minimizada. Quando essa minimização é bem-sucedida, as saídas da rede se aproximam dos valores-alvo, caracterizando uma rede treinada de forma satisfatória.

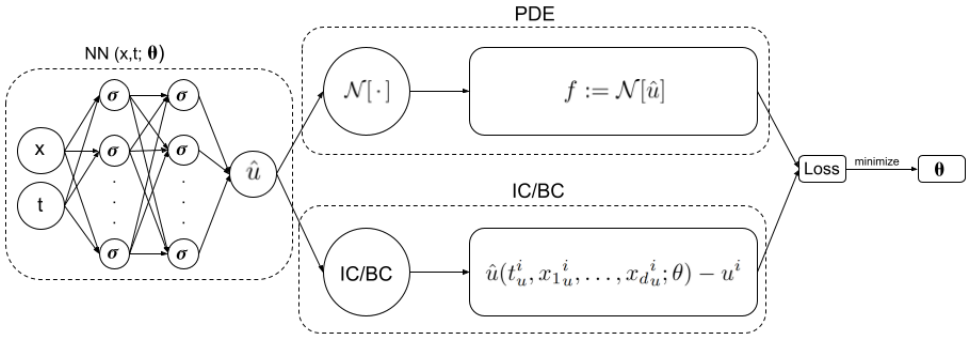
2.2 Redes neurais informadas pela física

Com o aumento dos recursos computacionais e a disponibilidade de dados, os modelos de aprendizado de máquina têm demonstrado grande robustez em diversas áreas, como reconhecimento de imagens e ciência cognitiva. No entanto, em áreas como Física, Biologia e Engenharias, onde os dados podem ser escassos e caros, os modelos tradicionais enfrentam desafios de convergência e generalização [1].

As *Physics-Informed Neural Networks* (PINNs), ou Redes Neurais Informadas pela Física, combinam o aprendizado profundo (*deep learning*) com princípios fundamentais das leis físicas, explorando simetrias, invariâncias e leis de conservação. Por meio da diferenciação automática, essas redes podem ajustar modelos complexos com poucos dados, integrando informações físicas ao processo de aprendizado [25].

Na Figura 6, é apresentado um fluxograma ilustrativo de uma PINN, destacando seu funcionamento como ferramenta computacional inovadora para modelar sistemas físicos.

Figura 6 – Fluxograma de uma PINN



2.2.1 Solução de equações diferencias parciais não lineares com redes neurais informadas pela física

O caso de interesse deste trabalho é a solução de equações diferenciais parciais não lineares orientada por dados, partindo de equações diferenciais bem definidas, KdV e Sine-Gordon, sendo proposta uma solução para as mesmas utilizando-se do método das PINNs.

De modo geral, o procedimento baseia-se no uso de uma rede neural totalmente conectada (*Fully Connected Neural Network*, FNN), o modelo mais simples de rede neural, também conhecido como perceptron multicamadas (*Multilayer Perceptron*, MLP), que é suficiente para a maioria dos problemas envolvendo equações diferenciais parciais (EDPs), conforme definido em [25].

Seja $\mathcal{F}^L(\mathbf{x}) : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ uma rede neural composta por L camadas, ou seja, uma rede com $(L-1)$ camadas ocultas, com N_ℓ neurônios na ℓ-ésima camada ($N_0 = d_{\text{in}}, N_L = d_{\text{out}}$). Seja $W^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ a matriz de pesos e $\mathbf{b}^\ell \in \mathbb{R}^{N_\ell}$ o vetor de *bias* da ℓ-ésima camada. Escolhendo-se uma função de ativação não linear σ, aplicada elemento a elemento, a FNN é definida recursivamente da seguinte forma:

$$
\begin{aligned}
&\text{Camada de entrada:} && \mathcal{F}^0(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^{d_{\text{in}}}, \\
&\text{Camadas ocultas:} && \mathcal{F}^\ell(\mathbf{x}) = \sigma(W^\ell \mathcal{F}^{\ell-1}(\mathbf{x}) + \mathbf{b}^\ell) \in \mathbb{R}^{N_\ell}, \quad 1 \leq \ell \leq L-1, \\
&\text{Camada de saída:} && \mathcal{F}^L(\mathbf{x}) = W^L \mathcal{F}^{L-1}(\mathbf{x}) + \mathbf{b}^L \in \mathbb{R}^{d_{\text{out}}},
\end{aligned}
$$

funções de ativação amplamente utilizadas incluem: a sigmoide logística $1/(1+e^{-x})$, a tangente hiperbólica (tanh) e a unidade linear retificada (*Rectified Linear Unit*, ReLU, definida como $\max\{x, 0\}$).

Nas PINNs, é essencial calcular as derivadas das saídas da rede neural em relação às suas entradas, o que pode ser realizado de maneira eficiente por meio de diferenciação automática (*Automatic Differentiation*, AD). Como a rede neural representa uma função composicional, a AD aplica repetidamente a regra da cadeia para calcular essas derivadas. O processo envolve duas etapas: uma passagem direta para calcular os valores de todas

as variáveis, seguida por uma passagem inversa para obter as derivadas correspondentes [25]. No exemplo ilustrado na Figura 6, observa-se tanto a arquitetura da rede neural $\mathbf{NN}(x,t;\theta)$ quanto a aplicação da diferenciação automática (AD), representada pela ação do operador diferencial não-linear $\mathcal{N}[\cdot]$.

As PINNs geralmente utilizam variáveis espaço-temporais (as variáveis independentes) como entradas, de modo que o número de neurônios na camada de entrada corresponde à dimensionalidade dessas variáveis. Após atravessarem as camadas ocultas, a camada de saída representa as variáveis dependentes, que podem ser um sistema de equações ou simplesmente a saída de uma função baseada nas variáveis de entrada. Nesse sentido, a rede neural pode ser vista como uma forma alternativa de representar uma função. A função utilizada para aproximar a solução $u(\mathbf{x})$ de uma equação diferencial não linear por meio de uma rede neural é representada como $\hat{u}(\mathbf{x};\boldsymbol{\theta})$, onde $\boldsymbol{\theta} = \{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{1\leq\ell\leq L}$ representa o conjunto de todas as matrizes de pesos e vetores de viés da rede neural $\hat{u}$.

Uma das principais vantagens das PINNs é sua flexibilidade na representação de soluções para equações diferenciais. Após o treinamento da rede neural $\hat{u}$ com pesos e *bias* $\boldsymbol{\theta}$ para aproximar a solução de $u(\mathbf{x})$, as PINNs são capazes de fornecer previsões contínuas de soluções de EDPs não lineares, mesmo quando treinadas com um número reduzido de pontos de dados discretos. Essa capacidade é alcançada ao incorporar restrições impostas por leis físicas, juntamente com condições iniciais e de contorno, permitindo um treinamento eficaz com dados limitados, uma característica essencial em situações onde dados experimentais são escassos ou dispendiosos de obter.

Neste contexto, em contraste com o que foi proposto em [1], cujo foco era o estudo de equações diferenciais parciais não lineares parametrizadas da forma:

$$u_t + \mathcal{N}[u;\lambda] = 0,\ x \in \Omega,\ t \in [0,T], \tag{2.1}$$

em que $u = u(t,x)$ se refere à solução latente oculta, $\mathcal{N}[\cdot;\lambda]$ a um operador não linear parametrizado por λ, e Ω é um subconjunto de $\mathbb{R}^D$; foi proposto, visando a possibilidade de se estudar equações com ordens superiores de derivação temporal, uma notação mais geral de parametrização de equações diferenciais não lineares, de forma similar ao apresentado em [25]. Tal mudança elucida a possibilidade de aplicação do método em diferentes contextos de parametrização de equações diferenciais, como por exemplo: ordens superiores de derivação temporal, ou ainda, casos independentes do tempo. O caso da equação de Sine-Gordon na subseção 2.5.2 conta com uma derivada temporal superior, sendo esta de segunda ordem.

Será considerado o tempo como uma componente especial do vetor de entrada $\mathbf{x}$, nos casos com dependência temporal, e também será considerado que $\boldsymbol{\Omega}$ contém o domínio temporal. Desta forma (2.1) poderia ser reescrita por:

$$\mathcal{N}[u;\lambda] = 0,\ \mathbf{x} \in \Omega. \tag{2.2}$$

Em que $u = u(\mathbf{x})$, com $\mathbf{x} = (t,x_1,...,x_d)$, se refere à solução latente oculta, $\mathcal{N}[\cdot;\lambda]$ a um operador não linear parametrizado por λ, e Ω é um subconjunto de $\mathbb{R}^D$, sendo D o número de dimensões espaciais mais a dimensão temporal, sendo esta considerada uma componente especial de $\mathbf{x}$. Por exemplo, no caso da equação de Sine-Gordon

$$\mathcal{N}[u;\lambda] = \lambda_1\, u_{tt} - \lambda_2\, u_{xx} + \lambda_3 \operatorname{sen}(u), \tag{2.3}$$

sendo $\lambda_1 = \lambda_2 = \lambda_3 = 1$, os valores usuais dos parâmetros da equação, e os índices subscritos em $\mathbf{u}$ representam derivadas parciais nas variáveis indicadas.

Como proposto em [1], o problema de se determinar quais parâmetros λ melhor representam os dados observados estaria ligado ao caso de descoberta de equações diferenciais parciais orientada por dados, e, para o caso em que os valores de λ sejam fixados, o objeto de estudo será o estado oculto $u(\mathbf{x})$, representando o caso de solução de equações diferenciais parciais orientada por dados.

Na abordagem usual das PINNs, para casos em que os parâmetros λ são fixados, a equação diferencial parcial genérica (2.1) pode ser reescrita como:

$$\mathcal{N}[u] = 0, \quad x,t \in \Omega, \tag{2.4}$$

em que $u = u(\mathbf{x})$, com $\mathbf{x} = (t,x_1,...,x_d)$, representa a solução latente oculta, $\mathcal{N}[\cdot]$ é um operador não-linear, e Ω denota o domínio espaço-temporal do problema. Este trabalho explora casos mais gerais de parametrização para equações diferenciais ordinárias e parciais não lineares, o que será discutido em detalhes nas Seções 3.1 e 3.3.

Em seguida é definida a função $f(\mathbf{x})$, chamada de função resíduo pelos autores de [1], em que $\mathbf{x} = (t,x_1,...,x_d)$ que representa o lado esquerdo da equação (2.4),

$$f := \mathcal{N}[u], \tag{2.5}$$

em que $u(\mathbf{x})$ será aproximado por uma rede neural profunda. Ao ser utilizado tanto esta suposição quanto a equação (2.5), será construída uma *rede neural informada pela física* (PINN) $f(\mathbf{x})$ que poderá ser diferenciada em relação a parâmetros da função aplicando a regra da cadeia fazendo uso da derivação automática [26], e possui os mesmos parâmetros da rede representada por $u(t,x)$, mesmo com diferentes funções de ativação, devido à ação do operador diferencial $\mathcal{N}$. Os parâmetros compartilhados entre $f(t,x)$ e $u(t,x)$ podem por sua vez ser aprendidos ao se minimizar o erro quadrático médio da função custo:

$$MSE = MSE_u + MSE_f. \tag{2.6}$$

sendo, com os termos explicitados

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_{1_u}^i, ..., x_{d_u}^i) - u^i|^2, \tag{2.7}$$

e

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_{1_f}, ..., x_{d_f}^i)|^2. \tag{2.8}$$

Aqui, o conjunto $[t_u^i, x_{1_u}^i, \ldots, x_{d_u}^i, u^i]_{i=1}^{N_u}$, em que N_u denota o número de pontos no *dataset* associado às condições de contorno, representa as condições iniciais e de contorno da função $u(t,x)$, podendo de forma mais geral depender de derivadas parciais de **u**, sendo avaliadas nas condições iniciais e de fronteira de u. Por sua vez, o conjunto $[t_f^i, x_{1_f}^i, \ldots, x_{d_f}^i]_{i=1}^{N_f}$, em que N_f indica o número de pontos de colocação, corresponde aos pontos utilizados na avaliação da equação diferencial $f(t,x)$, sendo tipicamente amostrados no interior do domínio $\mathbf{\Omega}$. Desta forma, o MSE_u corresponde às condições de contorno e de valor inicial, atuando de forma a garantir que as condições de fronteira sejam respeitadas. MSE_f por sua vez, tem o intuito de reger a estrutura imposta na equação (2.4), de forma a garantir que a rede encontrada represente a equação diferencial especificada.

Algo interessante a ser comentado é a ligação direta dos princípios da equação a ser estudada (que representa um problema físico) com a construção da rede neural. Ferramentas usuais de redes neurais profundas como diferenciação automática com respeito a parâmetros (*weight* e *bias*) da rede são utilizadas. A física é informada à rede neural, derivando o modelo com respeito às coordenadas de entrada (por exemplo, espaço e tempo), pelas exatas mesmas ferramentas, porém agora descrevendo equações diferenciais e condições de contorno.

O trabalho de se treinar estas redes neurais é feito com um número relativamente pequeno de dados (centenas até milhares) ao ser comparado com redes neurais que extraem informações unicamente dos dados (Por exemplo *MNIST Dataset*, 70.000 imagens com 28x28 *pixels* por imagem). Mesmo sem uma evidência teórica de que o método sempre convirja para um mínimo global, as evidências empíricas indicam uma boa precisão para equações diferenciais bem estabelecidas e com soluções únicas, ao ser utilizada uma rede neural suficientemente expressiva e com número suficiente de pontos de colocação N_f [1].

2.2.2 Implementação

Em [1] os autores utilizaram códigos programados em *python* fazendo uso da biblioteca *TensorFlow*, na versão 1 deste. A execução dos códigos, por se tratar de um modelo de *deep learning*, necessita de uma GPU, na maioria dos casos, para uma execução em tempo hábil. Neste trabalho a implementação foi realizada por meio do uso de GPUs feita via *Google Colab* [27]. Este disponibiliza gratuitamente, com limitações, GPUs para os

usuários em um ambiente de notebooks *Jupyter* totalmente online. Tal ambiente proporcionou a possibilidade de implementação dos códigos sem um investimento inicial de uma máquina potente o suficiente e com uma GPU dedicada.

Porém, algumas dificuldades foram enfrentadas: O *TensorFlow* ganhou uma nova versão, sendo esta a versão 2, o que, após um período de transição, fez a versão 1 deixar de ser suportada pelo *Google Colab*. Sem o suporte da versão 1, os códigos que utilizavam os mesmos para a implementação da PINN [1] deixaram de funcionar. Mesmo sendo possível a utilização de um modo de compatibilidade no *TensorFlow 2*, algumas bibliotecas, entre elas um dos otimizadores (L-BFGS-B), utilizados pelos autores de [1], pararam de ter uma implementação direta no modo compatibilidade do *TensorFlow 2*. Uma solução encontrada foi utilizar a biblioteca do python *DeepXDE*.

2.3 DeepXDE

DeepXDE [25] é uma biblioteca computacional desenvolvida em Python, projetada para fins educacionais e de pesquisa, aplicável à resolução de problemas de ciência e engenharia. A ferramenta é indicada para a resolução de equações diferenciais, incluindo casos não lineares, permitindo resolver problemas diretos a partir de condições iniciais e de contorno, bem como problemas inversos por meio da incorporação de medições adicionais. A biblioteca suporta domínios com geometria complexa utilizando a técnica de geometria sólida construtiva (CSG) e facilita a implementação de códigos compactos alinhados à formulação matemática. Este trabalho aborda a utilização e customização do DeepXDE, destacando sua eficácia e agilidade na implementação de PINNs, além da praticidade da ferramenta na resolução de EDPs.

A implementação do DeepXDE para resolver equações diferenciais parciais (EDPs) envolve etapas essenciais, como a definição do domínio computacional, a especificação da EDP, a aplicação das condições de contorno e iniciais, a construção e treinamento da rede neural, e a realização de predições. A seguir, descreve-se cada uma dessas etapas de forma detalhada.

2.3.1 Definição do Domínio Computacional

O DeepXDE define o domínio computacional por meio de seu módulo de geometria, permitindo especificar dimensões espaciais e temporais. Suporta geometrias primitivas, como `interval`, `rectangle` e `sphere`, além de composições via operações booleanas (`|`, `-`, `&`), utilizando Geometria Sólida Construtiva (CSG). Para problemas espaço-temporais, a classe `TimeDomain` estende o domínio com uma dimensão temporal.

A seguir, um exemplo de domínio unidimensional:

```
import deepxde as dde from deepxde.geometry import Interval

domain = Interval(0, 1) # Domínio de 0 a 1.
```

2.3.2 Definição da Equação Diferencial Parcial

A equação diferencial parcial é definida utilizando a gramática do TensorFlow. Como exemplo, a equação de Poisson

$$\nabla^2 u + 1 = 0$$

pode ser especificada no DeepXDE da seguinte forma:

```
def pde(x, y):
    dy_xx = dde.grad.hessian(y, x)
    return dy_xx + 1
```

Aqui, `dde.grad.hessian(y, x)` computa a segunda derivada de y em relação a x, ou seja, a diagonal do Hessiano, que no caso escalar corresponde à derivada segunda $\frac{d^2y}{dx^2}$.

2.3.3 Condições de Contorno e Iniciais

As condições de contorno e iniciais são essenciais para definir o problema. O DeepXDE oferece suporte a quatro condições de contorno padrão: Dirichlet (`DirichletBC`), Neumann (`NeumannBC`), Robin (`RobinBC`) e periódica (`PeriodicBC`), além de condições mais gerais via `OperatorBC`. Condições iniciais podem ser definidas com `IC`.

Por exemplo, uma condição de Dirichlet pode ser aplicada da seguinte forma:

```
def boundary(x, on_boundary):
    return on_boundary

bc = dde.DirichletBC(geometry=domain, func=lambda x: 0, on_boundary=boundary)
```

Aqui, a função `boundary` retorna o valor de `on_boundary`, que é um método já implementado no DeepXDE para verificar se um ponto pertence à fronteira do domínio. Assim, ao definir `on_boundary=boundary`, garante-se que a função especificada em `func`, que neste caso impõe o valor zero, seja aplicada exclusivamente nas bordas do domínio.

Para problemas dependentes do tempo, o DeepXDE também fornece a função `on_initial`, que verifica se um ponto pertence ao instante inicial do domínio temporal.

Isso permite a aplicação de condições iniciais de forma análoga às de contorno, garantindo que `func` seja imposta apenas no tempo inicial da simulação.

Outras condições, como Neumann, Robin e periódica, podem ser definidas de maneira semelhante, adaptando a função correspondente.

2.3.4 Combinação dos Componentes do Problema

Os componentes do problema, como a geometria, a EDP e as condições de contorno, são combinados em uma estrutura de dados. Para problemas independentes do tempo, utiliza-se:

```
data = dde.data.PDE(domain,
    pde,
    bcs=[bc],
    num_domain=100,
    num_boundary=20
)
```

Aqui, `num_domain` define o número de pontos dentro do domínio, distribuídos de forma pseudoaleatória. Já `num_boundary` especifica a quantidade de pontos amostrados aleatoriamente na fronteira do domínio.

Destaca-se que os pontos internos ao domínio não correspondem a pontos supervisionados no sentido clássico, pois são utilizados apenas como entradas para a aplicação da equação diferencial, definida em `pde`. Esses pontos permitem o cálculo das derivadas necessárias à solução do problema, funcionando como restrições implícitas impostas pela física do sistema. Por outro lado, os pontos das fronteiras e das condições iniciais podem ser considerados supervisionados, pois possuem valores conhecidos ou são explicitamente impostos pela equação diferencial nessas regiões.

De forma semelhante, para problemas dependentes do tempo, pode-se definir `num_initial`, que representa o número de pontos amostrados no instante inicial do domínio temporal.

2.3.5 Construção da Rede Neural

A rede neural é configurada utilizando o módulo `maps` do DeepXDE. Um exemplo de rede neural feedforward (FNN) com três camadas ocultas é:

```
net = dde.maps.FNN([1] + [20] * 3 + [1],
    activation="tanh",
```

```
    initializer="Glorot uniform"
)
```

Nesta configuração, a rede possui uma entrada única, correspondente à coordenada x previamente definida, três camadas ocultas com 20 neurônios cada e uma saída, que representa a função $f(x)$. A função de ativação `tanh` é aplicada apenas nas camadas ocultas, enquanto a camada de saída não possui ativação. Além disso, os pesos da rede são inicializados com o método `Glorot uniform`, garantindo uma distribuição adequada para o treinamento.

2.3.6 Definição e Compilação do Modelo

O modelo é construído combinando a definição da EDP com a arquitetura da rede neural:

```
model = dde.Model(data, net)
```

Durante a compilação do modelo, seleciona-se o otimizador e a taxa de aprendizado:

```
model.compile("adam", lr=1e-3)
```

Neste caso, o otimizador `adam` é escolhido, mas outros otimizadores, como o `L-BFGS` [28], também estão disponíveis para ajuste de hiperparâmetros durante o treinamento.

2.3.7 Treinamento e Realização de Predições

O treinamento da rede neural é realizado utilizando o método `train`, onde o número de épocas é especificado:

```
history = model.train(epochs=10000)
```

Após o treinamento, as soluções da EDP podem ser obtidas em diferentes pontos do domínio computacional utilizando o método `predict`:

```
solution = model.predict(X_test)
```

Destaca-se que a predição pode ser realizada em pontos arbitrários, caracterizando o modelo como mesh free — isto é, independente de uma malha fixa e capaz de ser avaliado em qualquer ponto do domínio. Ademais, o modelo não impõe restrições explícitas quanto

à realização de predições fora do domínio estudado. No entanto, na ausência de exemplos ou pontos de colocação que abranjam essas regiões externas durante o treinamento, a qualidade dos resultados tende a se degradar rapidamente.

Para mais exemplos e detalhes sobre a implementação, pode ser consultada a documentação e os tutoriais oficiais do DeepXDE.

2.3.8 Termos em Treinamento de Redes Neurais

Diversas técnicas são cruciais no treinamento de redes neurais. O otimizador **Adam (Adaptive Moment Estimation)** [29] é um método de gradiente descendente estocástico que adapta individualmente as taxas de aprendizado para cada parâmetro, utilizando estimativas de primeiro e segundo momentos dos gradientes. Frequentemente, é discutido em contextos mais amplos de otimização para aprendizado profundo [30].

A inicialização de pesos **Glorot Uniform**, também conhecida como Xavier Uniforme [31], visa manter a variância dos sinais propagados através da rede, sorteando pesos de uma distribuição uniforme cujos limites dependem das dimensões da camada. Este método auxilia na prevenção do desaparecimento ou explosão de gradientes [30].

O **L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno)** [32, 33] é um algoritmo de otimização pertencente à família dos métodos quasi-Newton, que aproxima a matriz Hessiana inversa usando um histórico limitado de gradientes e atualizações de parâmetros. É particularmente útil para otimizações em larga escala onde a Hessiana explícita é intratável [34].

Em seguida, são revisadas as equações diferenciais utilizadas no trabalho.

2.4 Equações Diferenciais

Uma Equação Diferencial Ordinária (EDO) é uma equação matemática que estabelece uma relação entre uma função desconhecida de uma única variável independente e uma ou mais de suas derivadas em relação a essa variável [35, 36]. O termo “ordinária” é utilizado para distinguir estas equações daquelas que envolvem derivadas parciais de funções com múltiplas variáveis independentes, conhecidas como Equações Diferenciais Parciais. Fundamentalmente, as EDOs são ferramentas matemáticas cruciais para descrever e modelar uma vasta gama de fenômenos e sistemas nos quais ocorrem variações e taxas de mudança. Elas permitem expressar de forma precisa como uma determinada quantidade evolui ou se comporta dinamicamente, baseando-se na maneira como suas próprias taxas de alteração instantânea se manifestam. Esta capacidade de encapsular a dinâmica de um sistema através de suas derivadas torna as EDOs indispensáveis em diversas áreas da ciência e engenharia.

Uma característica fundamental de uma EDO é a sua "ordem", que é definida pela ordem da mais alta derivada da função incógnita que aparece na equação [35, 36]. Por exemplo, a equação $y' = 2x$, onde y' representa a primeira derivada da função y em relação à variável x, é classificada como uma EDO de primeira ordem. De forma análoga, uma equação como $y'' + y = 0$, que envolve y'' (a segunda derivada de y), é um exemplo de uma EDO de segunda ordem. Equações como estas são encontradas na modelagem de inúmeros processos físicos e naturais, como a descrição do movimento de um objeto sob a influência de forças, a análise do crescimento ou decrescimento de populações, ou o estudo do decaimento radioativo de substâncias.

Resolver uma EDO consiste em encontrar uma função, ou uma família de funções, que, ao ser substituída na equação original juntamente com suas respectivas derivadas, satisfaça a equação, transformando-a em uma identidade matemática válida em um determinado intervalo da variável independente [36, 37]. O processo de resolução de uma EDO de ordem n geralmente conduz, inicialmente, a uma "solução geral". Esta solução geral é uma expressão que contém n constantes arbitrárias, representando uma família inteira de funções, cada uma das quais satisfaz a EDO [35, 36, 37]. O número dessas constantes está diretamente relacionado à ordem da EDO, refletindo o número de operações de integração inerentemente envolvidas na busca pela função original. Para se obter uma "solução particular", uma função específica que não contém constantes arbitrárias e que descreve um cenário particular, são necessárias informações adicionais. Estas informações são tipicamente fornecidas na forma de "condições iniciais" (ou, em alguns contextos, condições de contorno), que permitem determinar os valores específicos das constantes presentes na solução geral [35, 36, 37].

Nesta seção, são apresentados os métodos de solução para equações diferenciais de segunda ordem, além de conceitos fundamentais sobre condições de contorno aplicadas a equações diferenciais parciais. As equações diferenciais não lineares associadas a sólitons serão abordadas separadamente na Seção 2.5.

2.4.1 Equação Diferencial de Segunda Ordem

Considere a equação diferencial linear de segunda ordem homogênea com coeficientes constantes:

$$a\frac{d^2y}{dt^2} + b\frac{dy}{dt} + cy = 0, \tag{2.9}$$

onde a, b e c são constantes reais, com $a \neq 0$. Dividindo a equação por a, obtém-se:

$$\frac{d^2y}{dt^2} + A\frac{dy}{dt} + By = 0, \tag{2.10}$$

com $A = \frac{b}{a}$ e $B = \frac{c}{a}$. A solução geral depende das raízes da equação característica:

$$r^2 + Ar + B = 0. \tag{2.11}$$

As raízes são dadas por:

$$r = \frac{-A \pm \sqrt{A^2 - 4B}}{2}. \tag{2.12}$$

O comportamento da solução depende do discriminante $\Delta = A^2 - 4B$.

2.4.1.1 Aplicações

Essa equação modela diversos fenômenos físicos, como o movimento harmônico amortecido, descrito por:

$$m\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + kx = 0, \tag{2.13}$$

onde m é a massa, γ é o coeficiente de amortecimento, e k é a constante elástica. A solução descreve sistemas que podem ser:

- **Subamortecidos** ($\Delta < 0$): oscilações com amplitude decrescente.
- **Criticamente amortecidos** ($\Delta = 0$): retorno ao equilíbrio sem oscilar.
- **Superamortecidos** ($\Delta > 0$): retorno lento ao equilíbrio sem oscilar.

Esses casos estão diretamente relacionados às soluções da equação diferencial, conforme discutido a seguir.

2.4.1.2 Solução Geral

A solução geral de uma equação diferencial homogênea de segunda ordem depende das raízes da sua equação característica. Essas raízes são determinadas pelo discriminante (Δ) da equação característica, que por sua vez depende dos coeficientes A e B da equação diferencial. A natureza das raízes (reais e distintas, reais e iguais, ou complexas conjugadas) define a forma da solução geral [38]. Abaixo, detalhamos cada um dos casos possíveis:

Caso 1: Raízes Reais e Distintas ($\Delta > 0$)

Quando o discriminante é positivo ($\Delta > 0$), a equação característica possui duas raízes reais e distintas, r_1 e r_2. Nesse caso, a solução geral da equação diferencial é uma combinação linear de duas exponenciais, cada uma correspondendo a uma das raízes:

$$y(x) = C_1 e^{r_1 x} + C_2 e^{r_2 x}, \tag{2.14}$$

onde C_1 e C_2 são constantes determinadas pelas condições iniciais do problema. Esse caso é comum em sistemas superamortecidos, onde não há oscilações.

Caso 2: Raízes Reais e Iguais ($\Delta = 0$)

Quando o discriminante é zero ($\Delta = 0$), a equação característica possui uma raiz real dupla, $r = -\frac{A}{2}$. Nesse caso, a solução geral inclui um termo adicional multiplicado por x para garantir que as soluções sejam linearmente independentes:

$$y(x) = (C_1 + C_2 x)e^{-\frac{Ax}{2}}. \tag{2.15}$$

Aqui, C_1 e C_2 também são constantes determinadas pelas condições iniciais. Esse caso é típico de sistemas criticamente amortecidos, onde o sistema retorna ao equilíbrio no menor tempo possível sem oscilar.

Caso 3: Raízes Complexas Conjugadas ($\Delta < 0$)

Quando o discriminante é negativo ($\Delta < 0$), as raízes da equação característica são complexas conjugadas, da forma $r = \alpha \pm i\beta$, onde $\alpha = -\frac{A}{2}$ e $\beta = \frac{\sqrt{4B-A^2}}{2}$. Nesse caso, a solução geral envolve funções trigonométricas (seno e cosseno) que descrevem oscilações amortecidas:

$$y(x) = e^{-\frac{Ax}{2}} \left(C_1 \cos\left(\frac{\sqrt{4B - A^2}}{2} x \right) + C_2 \sin\left(\frac{\sqrt{4B - A^2}}{2} x \right) \right). \tag{2.16}$$

Aqui, C_1 e C_2 são constantes que dependem das condições iniciais. Esse caso é característico de sistemas subamortecidos, onde o sistema oscila enquanto retorna ao equilíbrio.

2.4.1.3 Solução Hiperbólica

A solução geral pode ser expressa de forma unificada usando funções hiperbólicas. Para isso, partimos da solução no caso de raízes reais e distintas ($\Delta > 0$):

$$y(x) = e^{-\frac{Ax}{2}} \left(C_1 e^{\frac{\sqrt{A^2-4B}}{2}x} + C_2 e^{-\frac{\sqrt{A^2-4B}}{2}x} \right). \tag{2.17}$$

Definindo $z = \frac{\sqrt{A^2-4B}}{2}x$, podemos reescrever a solução como:

$$y(x) = e^{-\frac{Ax}{2}} \left(C_1 e^{z} + C_2 e^{-z} \right). \tag{2.18}$$

Usando as definições das funções hiperbólicas:

$$\cosh(z) = \frac{e^z + e^{-z}}{2}, \quad \sinh(z) = \frac{e^z - e^{-z}}{2}, \tag{2.19}$$

podemos expressar e^z e e^{-z} como:

$$e^z = \cosh(z) + \sinh(z), \quad e^{-z} = \cosh(z) - \sinh(z). \tag{2.20}$$

Substituindo em (2.18), obtemos:

$$y(x) = e^{-\frac{Ax}{2}} \left(C_1(\cosh(z) + \sinh(z)) + C_2(\cosh(z) - \sinh(z))\right). \tag{2.21}$$

Agrupando os termos, temos:

$$y(x) = e^{-\frac{Ax}{2}} \left((C_1 + C_2)\cosh(z) + (C_1 - C_2)\sinh(z)\right). \tag{2.22}$$

Escolhendo $C_1 + C_2 = 1$ e $C_1 - C_2 = \frac{A}{\sqrt{A^2-4B}}$, obtemos a solução hiperbólica:

$$y(x) = e^{-\frac{Ax}{2}} \left(\cosh(z) + \frac{A}{\sqrt{A^2 - 4B}}\sinh(z)\right). \tag{2.23}$$

Essa forma é válida para todos os casos, com interpretações adequadas para $\Delta = 0$ e $\Delta < 0$.

A seguir, é apresentado os sólitons e a equação KdV.

2.4.2 Equações Diferenciais Parciais

As Equações Diferenciais Parciais (EDPs) são ferramentas matemáticas essenciais para descrever uma miríade de fenômenos físicos, desde a propagação do calor até a dinâmica dos fluidos [39, 40, 41]. Diferentemente de muitas Equações Diferenciais Ordinárias (EDOs), uma EDP por si só raramente define uma solução única. Em vez disso, ela descreve uma família de funções que satisfazem a relação diferencial. Para particularizar uma solução que corresponda a um cenário físico específico, informações adicionais na forma de condições auxiliares são indispensáveis. Estas são categorizadas como Condições Iniciais (CIs), especificando o estado do sistema em um tempo inicial (tipicamente $t = 0$), e Condições de Contorno (CCs), que ditam o comportamento da solução nas fronteiras do domínio espacial em que o problema é definido [40, 42]. A escolha e o número corretos dessas condições são cruciais para garantir não apenas a unicidade da solução, mas também sua relevância física e estabilidade, como será discutido.

2.4.3 Determinação do Número de Condições

Condições Iniciais (CIs): Para EDPs evolutivas (i.e., dependentes do tempo), o número de condições iniciais necessárias é determinado pela ordem m da derivada temporal mais alta presente na equação. Ou seja, se a EDP envolve até a m-ésima derivada em

relação ao tempo t, então são necessárias m condições iniciais prescritas sobre a variável dependente u e suas derivadas temporais até ordem $m-1$, tipicamente especificadas em uma superfície inicial $t = t_0$ (mas não necessariamente $t = 0$) [43].

Condições de Contorno (CCs): A quantidade e o tipo de condições de contorno exigidas dependem da ordem e natureza das derivadas espaciais na EDP, bem como da geometria do domínio. Para equações de ordem espacial n, frequentemente são exigidas até n condições em cada ponto da fronteira onde o problema for definido, embora o número exato dependa da forma e tipo da equação (elíptica, parabólica ou hiperbólica). Estas condições podem incluir valores da função (u), suas derivadas normais ($\partial u/\partial \nu$), ou combinações lineares das mesmas.

Tipos comuns de condições:

- **Dirichlet:** Prescreve o valor de u em uma parte da fronteira.
- **Neumann:** Prescreve a derivada normal de u na fronteira.
- **Robin (mista):** Prescreve uma combinação linear entre u e $\partial u/\partial \nu$.
- **Cauchy:** Prescreve simultaneamente u e $\partial u/\partial \nu$ (ou outras derivadas) sobre uma mesma superfície. Para EDPs evolutivas, essa superfície é geralmente uma hipersuperfície não característica no espaço-tempo, como $t = t_0$, onde se fornecem os dados iniciais [43, 44].

2.4.4 Problemas bem-postos (Well-Posedness)

Um problema envolvendo uma EDP e suas condições auxiliares é considerado **bem-posto** (traduzindo do inglês *well-posed*) se satisfaz os três critérios de Hadamard [39, 45]:

1. **Existência:** Uma solução para o problema deve existir.
2. **Unicidade:** A solução deve ser única.
3. **Estabilidade (Dependência Contínua dos Dados):** A solução deve depender continuamente dos dados fornecidos. Pequenas perturbações nas CIs, CCs ou termos não homogêneos devem resultar em pequenas alterações na solução.

A especificação correta do número e tipo de CIs/CCs é fundamental. Condições insuficientes geralmente levam à não unicidade, enquanto condições excessivas podem impedir a existência de uma solução. Condições inadequadas ao tipo de EDP podem levar à instabilidade (e.g., resolver a equação do calor retroativamente no tempo é um problema notoriamente mal posto por violar a estabilidade) [46, 43].

Problemas bem-postos em EDPs Não Lineares e Caos

Para EDPs não lineares, a questão da boa colocação pode ser particularmente complexa. Muitas EDPs não lineares, como as equações de Navier-Stokes na dinâmica dos fluidos ou equações de reação-difusão em biologia química, podem exibir comportamento **caótico** sob certas condições e regimes de parâmetros [47, 48]. O caos é caracterizado por uma sensibilidade extrema às condições iniciais. Isso significa que duas trajetórias de solução que começam arbitrariamente próximas no espaço de fase divergirão exponencialmente ao longo do tempo [49].

Do ponto de vista de Hadamard:

- **Existência e Unicidade (curto prazo):** Para muitas EDPs não lineares caóticas, pode-se ainda estabelecer a existência e unicidade de soluções por um período de tempo finito (localmente no tempo).
- **Estabilidade (longo prazo):** A sensibilidade extrema às CIs implica que o terceiro critério de Hadamard (dependência contínua no sentido de estabilidade para previsões de longo prazo) é violado de forma prática. Embora a solução possa depender continuamente dos dados em um sentido topológico para tempos finitos, qualquer pequena incerteza ou erro nos dados iniciais será amplificado exponencialmente, tornando a previsão precisa do estado individual do sistema a longo prazo impossível.

Assim, para sistemas caóticos descritos por EDPs não lineares, enquanto o problema pode ser bem posto para soluções de curto prazo, a previsibilidade a longo prazo é perdida. A análise de tais sistemas frequentemente se volta para descrições estatísticas do comportamento, o estudo de atratores (incluindo atratores estranhos que caracterizam o caos) e médias de conjunto, em vez de buscar soluções de trajetória única a longo prazo [47, 49]. A pesquisa sobre a boa colocação para EDPs como Navier-Stokes (um dos problemas do Millennium Prize) ainda é uma área ativa, especialmente em relação à regularidade global de soluções para dados suaves.

2.4.4.1 Problemas bem-postos e PINNs

Ao se utilizar um modelo de redes neurais para resolver equações diferenciais, propõe-se uma abordagem consideravelmente distinta das técnicas numéricas clássicas.

As condições de contorno, por exemplo, não são impostas de forma rígida, mas fornecidas à rede como exemplos considerados verdadeiros, ou ainda, como imposições físicas que ela deve respeitar em pontos específicos do domínio. Esses exemplos são incorporados ao processo de treinamento por meio da função custo, o que confere à rede maior flexibilidade na forma como tais condições são tratadas.

Essa característica abre espaço para aplicações que extrapolam o escopo tradicional dos métodos numéricos, como o uso de condições de contorno não usuais ou distribuídas de maneira assimétrica. Ainda que não seja possível prever com certeza como a rede se comportará em cenários ill-posed, este trabalho explora algumas dessas situações, assim como [50], com foco na análise das predições obtidas. Os resultados sugerem que, mesmo em condições que fogem dos critérios clássicos de "bem-postos", as PINNs podem oferecer soluções coerentes. Isso se mostra especialmente interessante para problemas inversos, uso de técnicas de regularização e abordagens com resolução contínua, tópicos que permanecem abertos para investigações futuras.

2.5 Sólitons

"Sólitons" foram inicialmente observados e documentados por J. Scott Russell em 1834 [51]:

> *"I was observing the motion of a boat which was rapidly drawn along a narrow channel by a pair of horses, when the boat suddenly stopped — not so the mass of water in the channel which it had put in motion; it accumulates round the prow of the vessel in a state of violent agitation, then suddenly leaving it behind, rolled forward with great velocity, assuming the form of a large solitary elevation, a rounded, smooth and well- defined heap of water, which continued its course along the channel apparently without change of form or diminution of speed. I followed it on horseback, and overtook it still rolling on at a rate of some eight or nine miles an hour, preserving its original figure some thirty feet long and a foot to a foot and a half in height. Its height gradually diminished, and after a chase of one or two miles I lost it in the windings of the channel. Such, in the month of August 1834, was my first chance interview with that rare and beautiful phenomenon which I have called the Wave of Translation ... (RUSSEL, 1834)."* [1]

Segundo [5], a partir desta observação experimentos foram realizados em laboratório com tanques d'água com profundidade constante e entre as conclusões surgiram as seguintes observações:

1. A existência de ondas solitárias, longas e rasas que possuem forma permanente.

[1] "Eu estava observando o movimento de um barco que foi rapidamente puxado ao longo de um canal estreito por um par de cavalos, quando o barco parou de repente - não só a massa de água no canal foi colocada em movimento; acumula-se à volta da proa do navio numa estado de agitação violenta, deixando-o subitamente para trás, rolou para a frente com grande velocidade, assumindo a forma de uma grande elevação solitária, arredondada, lisa e bem amontoado de água definido, que continuou seu curso ao longo do canal aparentemente sem mudança de forma ou diminuição da velocidade. Eu o segui a cavalo e o ultrapassei ainda rolando a uma velocidade de cerca de oito ou nove milhas por hora, preservando sua figura original cerca de trinta pés de comprimento e um pé a um pé e meio de altura. Sua altura gradualmente diminuiu, e depois de uma perseguição de uma ou duas milhas eu o perdi nas sinuosidades do canal. Tal foi, no mês de agosto de 1834, meu primeira contato casual com aquele raro e belo fenômeno que chamei de Onda de Translação... (RUSSEL, 1834)"

2. Uma forma de relacionar a velocidade de propagação **c** com a aceleração da gravidade **g**, a profundidade constante do tanque **h** e a amplitude da onda $\boldsymbol{\eta}$

$$c^2 = g(h + \eta). \tag{2.24}$$

Outras investigações foram realizadas, em seguida, por Airy, Stokes, Boussinesq e Rayleigh [5]. Os pesquisadores Boussinesq e Rayleigh encontraram descrições aproximadas independentemente, e Boussinesq derivou uma solução unidimensional não-linear para o problema, equação que possui seu nome atualmente.

Estas investigações geraram controvérsias a respeito da possibilidade de equações de ondas não-viscosas de água apresentarem soluções do tipo ondas solitárias. Esta questão foi resolvida por Korteweg e de Vries em 1895 [52].

Em uma definição mais precisa, faz se uma distinção entre ondas solitárias e sólitons, estes são menos gerais possuindo algumas condições extras, além de serem ondas solitárias.

Segundo [5] um sóliton é uma onda solitária que assintoticamente preserva o seu formato e velocidade mediante interação não linear com outras ondas solitárias, ou de forma mais geral, com outro distúrbio localizado.

Ainda em relação a sua forma, existe na literatura [53] a distinção, sendo possível identificar sólitons do tipo *lump* e *kink*, sendo os sólitons da equação KdV estudados neste trabalho do tipo *lump*, e os sóliton estudado na equação Sine-Gordon do tipo *kink*.

Os sólitons do tipo *lump* são soluções de ondas solitárias que emergem tipicamente em equações diferenciais parciais não lineares, como Kadomtsev-Petviashvili e KdV. Matematicamente, eles são caracterizados por serem funções racionais das coordenadas espaciais, o que lhes confere uma localização em todas as direções e um decaimento algébrico para zero à medida que se afasta do seu centro. Os *lumps* representam "caroços" de energia ou perturbação que se movem sem se dispersar, mas sua estabilidade é de natureza dinâmica, intrinsecamente ligada às propriedades específicas da equação que os governa, e geralmente não possuem uma carga topológica associada.

Por outro lado, os sólitons do tipo *kink* são fundamentalmente estruturas topológicas, mais comumente encontradas em sistemas unidimensionais descritos por teorias de campo não lineares, como a equação de Sine-Gordon ou a teoria ϕ^4. Um kink representa uma transição estável e suave entre dois estados de vácuo (ou equilíbrio) distintos, mas energeticamente equivalentes, do sistema. Assim, a solução do campo não decai para zero em ambas as direções, mas assimptota a valores constantes diferentes em $x \to \pm\infty$. Sua notável estabilidade não se deve primariamente à integrabilidade da equação, mas à conservação de uma "carga topológica", que impede o *kink* de se desfazer ou transicionar

para um estado de vácuo uniforme. Eles são interpretados fisicamente como paredes de domínio, interfaces ou defeitos topologicamente protegidos.

2.5.1 Equação de Korteweg-De Vries

A equação de Korteweg–De Vries ou KdV foi derivada de uma equação não linear que governava a evolução de ondas longas de uma dimensão, de pequena amplitude, sob o efeito da gravidade superficial e que se propagavam em canais rasos de água. Ela é dada por

$$\frac{\partial \eta}{\partial \tau} = \frac{3}{2}\sqrt{\frac{g}{h}}\frac{\partial}{\partial \xi}\left(\frac{1}{2}\eta^2 + \frac{2}{3}\alpha\eta + \frac{1}{3}\sigma\frac{\partial^2\eta}{\partial \xi^2}\right), \qquad \sigma = \frac{1}{3}h^3 - Th/(\rho g), \tag{2.25}$$

onde $\boldsymbol{\eta}$ representa a elevação da superfície acima do nível de equilíbrio h, $\boldsymbol{\alpha}$ é uma constante pequena arbitrária relacionada ao movimento uniforme do líquido, $\mathbf{g}$ a aceleração da gravidade, $\mathbf{T}$ representa a tensão superficial e $\boldsymbol{\rho}$ representa a densidade. A equação (2.25) é a equação de Korteweg-De Vries, que apresenta soluções de ondas permanentes, incluindo soluções de ondas solitárias.

A equação pode ser transformada para uma forma adimensional usando as transformações de coordenadas

$$t = \frac{1}{2}\sqrt{g/(h\sigma)}\tau, \qquad x = -\sigma^{-\frac{1}{2}}\xi, \qquad u = \frac{1}{2}\eta + \frac{1}{3}\alpha. \tag{2.26}$$

Ao substituir (2.26) em (2.25) obtém-se

$$u_t + 6uu_x + u_{xxx} = 0. \tag{2.27}$$

Essa será a forma da equação KdV que se utilizará ao longo deste trabalho.

Apesar de ser uma equação diferencial parcial não linear, uma solução exata do tipo de onda solitária pode ser encontrada de forma relativamente simples usando o procedimento de soluções de ondas viajantes com uma transformação de coordenadas do tipo

$$u(t,x) = z(x - \beta t) \equiv z(\zeta), \tag{2.28}$$

onde $\boldsymbol{\beta}$ uma constante a ser determinada, representa uma velocidade constante de deslocamento da onda solitária. Pode-se aplicar (2.28) em (2.27) e reescrever a equação como uma equação ordinária:

$$-\beta\,\frac{d\,z}{d\,\zeta} + 6\,z\,\frac{d\,z}{d\,\zeta} + \frac{d^3\,z}{d\,\zeta^3}. \tag{2.29}$$

Por integração direta, utilizando c_1 para denotar a constante de integração

$$-\beta\,z + 3\,z^2 + \frac{d^2\,z}{d\,\zeta^2} = c_1, \tag{2.30}$$

multiplicando ambos os lados da equação por $\frac{d\,z}{d\,\zeta}$

$$\begin{aligned} -\beta\, z\,\frac{d\,z}{d\,\zeta} + 3\,z^2\,\frac{d\,z}{d\,\zeta} + \frac{d^2\,z}{d\,\zeta^2}\,\frac{d\,z}{d\,\zeta} &= c_1\,\frac{d\,z}{d\,\zeta} \\ \implies -\beta\, z\, d\,z + 3\,z^2\, d\,z + \frac{d^2\,z}{d\,\zeta^2}\, d\,z &= c_1\, d\,z, \end{aligned} \tag{2.31}$$

observando que por uma regra da cadeia $\frac{1}{2}\frac{d}{d\,\zeta}(\frac{d\,z}{d\,\zeta})^2 = \frac{d\,z^2}{d\,\zeta^2}$, integrando novamente e utilizando c_2 como segunda constante de integração:

$$-\frac{\beta}{2}z^2 + z^3 + \frac{1}{2}\left(\frac{d\,z}{d\,\zeta}\right)^2 = c_1\,z + c_2. \tag{2.32}$$

Resta uma equação diferencial de primeira ordem a ser resolvida e que está em função de duas constantes arbitrárias c_1 e c_2. Procurando uma solução particular em que $c_1 = c_2 = 0$ que pode ser exigida se for imposta a condição em que, para $x \to \pm\infty$ deve-se ter $z \to 0$, $\frac{dz}{d\zeta} \to 0$, $\frac{d^2z}{d\zeta^2} \to 0$. A equação (2.32) pode ser simplificada da forma:

$$\left(\frac{d\,z}{d\,\zeta}\right)^2 = z^2(\beta - 2z) \tag{2.33}$$

Soluções mais gerais podem ser obtidas sem as condições impostas na equação (2.32), mas atentando no momento para esta simplificação, é possível resolver a equação (2.33) por separação de variáveis:

$$\frac{d\,z}{z\sqrt{(\beta - 2\,z)}} = d\,\zeta \tag{2.34}$$

Com uma substituição conveniente de variáveis da forma

$$z := \frac{1}{2}\beta(1 - sech^2\, w), \tag{2.35}$$

usando a relação $cosh^2\, w - sinh^2\, w = 1$

$$d\,z = -\beta\frac{senh\, w}{cosh^3\, w}\, d\,w. \tag{2.36}$$

Ainda, observando que de (2.35) tem-se a relação:

$$w = sech^{-1}\sqrt{\frac{2\,z}{\beta}}. \tag{2.37}$$

Com ajuda das equações (2.35, 2.36, 2.37) é possível resolver a equação (2.34) para z, retornando após para as variáveis originais:

$$u(t,x) = \frac{\beta}{2}\, sech^2\left[\frac{\sqrt{\beta}}{2}(x - \beta\, t)\right] \tag{2.38}$$

É notado na equação (2.38) que diferentemente de uma equação de onda padrão, existe uma relação direta entre a amplitude e a velocidade de propagação da onda. É notado que uma onda com maior amplitude terá uma maior velocidade de propagação.

2.5.1.1 Leis de Conservação da Equação KdV

De acordo com [5], a equação de Korteweg-De Vries (KdV) possui um número infinito de leis de conservação independentes, o que foi um avanço crucial na solução dessa equação. Para uma equação diferencial parcial

$$\Delta(x, t, u(x, t)) = 0, \tag{2.39}$$

uma lei de conservação é dada por:

$$D_t T + D_x X = 0, \tag{2.40}$$

onde $T(x, t)$ é a densidade conservada e $X(x, t)$ é o fluxo associado. Se $u \to 0$ conforme $|x| \to \infty$, então a integral de T sobre todo o espaço é constante no tempo, indicando uma quantidade conservada.

Para a equação KdV (2.27), as primeiras três leis de conservação são dadas por:

1.
$$(u)_t + (3u^2 + u_{xx})_x = 0 \quad \text{(Conservação do momento)}, \tag{2.41}$$

2.
$$(u^2)_t + (4u^3 + 2uu_{xx} - u_x^2)_x = 0 \quad \text{(Conservação da energia)}, \tag{2.42}$$

3.
$$\left(u^3 - \frac{1}{2}u_x^2\right)_t + \left(\frac{9}{2}u^4 + 3u^2u_{xx} - 6uu_x^2 - u_x u_{xxx} + \frac{1}{2}u_{xx}^2\right)_x = 0, \tag{2.43}$$

em que os termos subscritos representam derivadas parciais em relação a variáveis representadas.

Estas leis de conservação são fundamentais para a análise das soluções da equação KdV, fornecendo invariantes que ajudam a entender a dinâmica e a estabilidade das ondas do tipo sóliton, características das equações integráveis.

Assim como a equação de KdV, outras equações possuem soluções do tipo sóliton, sendo outro exemplo a equação Sine-Gordon.

2.5.2 Equação Sine-Gordon

Segundo [53] a equação de Sine-Gordon

$$\varphi_{tt} - \varphi_{xx} + \operatorname{sen}\varphi = 0, \tag{2.44}$$

possui este nome devido a comparações utilizando a presença do termo seno (*sine*) e a equação de Klein-Gordon:

$$\varphi_{tt} - \varphi_{xx} + \varphi = 0 \tag{2.45}$$

podendo a equação (2.44) ser vista como uma contraparte não linear da equação (2.45) [6].

A equação de Sine-Gordon também pode ser representada da forma

$$\varphi_{uv} = \sin\varphi, \tag{2.46}$$

ao se utilizar as cordenadas do "cone de luz" (u, v)

$$u = \frac{x+t}{2}, \quad v = \frac{x-t}{2}. \tag{2.47}$$

De acordo com [6], a equação foi descoberta pela primeira vez na forma (2.46), pelo matemático francês Jacques Edmond Émile Bour em 1862, como a equação de Gauss-Codazzi descrevendo superfícies bidimensionais com curvatura negativa constante.

A equação é integrável, possuindo uma vasta família de soluções exatas, descobertas no século 19 por meio de transformação de Bäcklund [54]. Possuindo ainda aplicações na forma do limite contínuo de Frenkel-Kontorova (FK), *long Josephson junction*, transparência auto-induzida (SIT). Na física do estado sólido, a equação e suas versões estendidas encontram várias aplicações como modelos de ondas de spin em meios magnéticos; entre outras aplicações.

Utilizando a transformada de Backlünd, na equação (2.46), faz-se com que φ se relacione com φ' por meio de equações de primeira ordem

$$\begin{aligned} \varphi'_u &= \varphi_u + 2\beta \sin\frac{\varphi' + \varphi}{2} \\ \varphi'_v &= -\varphi_v + \frac{2}{\beta}\sin\frac{\varphi' - \varphi}{2}, \end{aligned} \tag{2.48}$$

sendo β um parâmetro real arbitrário, de forma que, dado uma solução φ para a equação de Sine-Gordon (2.46), φ' também será uma solução. Desta forma, utilizando a solução trivial $\varphi = \varphi_0 = 0$ em (2.48) é obtido

$$\begin{aligned} \varphi'_u &= 2\beta \sin\frac{\varphi'}{2} \\ \varphi'_v &= \frac{2}{\beta}\sin\frac{\varphi'}{2}, \end{aligned} \tag{2.49}$$

a equação (2.49) por sua vez pode ser integrada diretamente para chegar na solução de um sóliton tipo *kink* que, ao se retornar para coordenadas habituais, se tornam

$$\varphi_{\text{soliton}}(x,t) := 4\arctan\left(e^{\gamma(x-vt)+\delta}\right), \tag{2.50}$$

sendo $\boldsymbol{v}$ a velocidade de propagação da onda, que se relaciona com o parâmetro arbitrário real β, $\boldsymbol{\delta}$ é um fator de fase de onda, e por último, $\boldsymbol{\gamma}$ representa o fator de Lorentz em unidades de $\mathbf{c}$=1

$$\gamma^2 = \frac{1}{1-v^2}. \tag{2.51}$$

Algo interessante a ser pontuado é que a auto-transformada de Backlünd gera a solução tipo um sóliton com uma solução trivial sendo aplicada a (2.48). Se uma solução do tipo sóliton for aplicada para φ, soluções do tipo vários sólitons podem ser obtidas sucessivamente.

Utilizando os mesmos passos, uma equação de Sine-Gordon mais geral pode ser descrita por

$$\varphi_{tt} - \varphi_{xx} + m^2 \sin \varphi = 0, \tag{2.52}$$

e possui uma solução do tipo sóliton (kink) dada por

$$\varphi_{\text{soliton}}\,(x,t) := 4 \arctan\left(e^{m\,\gamma(x-vt)+\delta}\right). \tag{2.53}$$

3 Metodologia

Neste Capítulo são apresentadas as metodologias adotadas no estudo, sendo explicitados testes com uma equação diferencial de segunda ordem com coeficientes variáveis com o intuito de apresentar o método e testar a capacidade de generalização de uma equação diferencial ordinária com coeficientes variáveis. Em seguida, será apresentada a metodologia adotada para os experimentos envolvendo a equação KdV e a equação Sine-Gordon. Inicialmente, descrevem-se os procedimentos e configurações utilizados para estruturar os testes, incluindo a definição das arquiteturas de rede a serem avaliadas na equação KdV. Posteriormente, discute-se a abordagem empregada na equação Sine-Gordon, com foco na análise da generalização por meio da variação de parâmetros da equação diferencial.

As implementações foram desenvolvidas em Python no formato .ipynb e estão disponíveis no repositório do GitHub: <https://github.com/igoraroliveira/MSc>

3.1 Equação diferencial ordinária de segunda ordem com coeficientes variáveis

Uma abordagem pouco explorada nas PINNs é a generalização de equações diferenciais com parâmetros variáveis. Neste estudo, é testada a abordagem das PINNs em uma equação diferencial de segunda ordem simples, que possui dois coeficientes variáveis. Esta pesquisa tem como objetivo avaliar o desempenho do método em diferentes contextos, destacando sua adaptabilidade a equações diferenciais caracterizadas por uma família de soluções, comuns em equações com coeficientes variáveis.

3.1.1 Elucidando o problema

Uma equação de seguinte forma:

$$y''(x) + A\,y'(x) + B\,y(x) = 0, \tag{3.1}$$

usualmente, com os parâmetros A e B fixos, representa uma equação ordinária (E.D.O.) de segunda ordem, com uma variável x e que possui solução trivial que depende dos parâmetros A e B por meio da equação característica da E.D.O., ou ainda, uma família de soluções, que pode ser fixada ao se especificar condições de contorno para o problema.

Porém, o objetivo será analisar se uma rede neural pode generalizar as soluções da equação diferencial no caso em que o domínio da função e os intervalos possíveis das

variáveis foram definidos como:

$$A \in [0, 1], \quad B \in [0, 1], \quad x \in [0, 1.58], \tag{3.2}$$

com as condições iniciais:

$$y(0) = 1, \quad y'(0) = 0 \tag{3.3}$$

A capacidade de generalização de uma rede neural para resolver a equação (3.1) foi testada, onde ambos os coeficientes A e B variavam nos intervalos definidos. As condições iniciais foram definidas como na equação (3.3). É importante destacar que tais condições iniciais serão definidas de forma única para a equação (3.1) para os intervalos dos parâmetros e variáveis representados na equação (3.2), sendo as condições independentes dos parâmetros A e B.

3.1.2 Implementação

Utilizando a biblioteca *deepxde* [25], a equação diferencial (3.1) foi representada, correspondendo à função residual do problema como definido em [1]. Essa função será introduzida na rede neural como uma função custo (*loss function*). O trecho do código que representa esta passagem vem dado a seguir:

```
def pde(x, y):
    dy_x = dde.grad.jacobian(y, x,j=2)
    dy_xx = dde.grad.hessian(y, x,i=2,j=2)

    A = x[:,0:1]
    B = x[:,1:2]

    return dy_xx + A * dy_x + B * y
```

Uma vez que se é buscada uma generalização não para um valor fixo de A e B, mas sim para um intervalo de valores, a forma encontrada para se representar tal problema foi a de se adicionar entradas extras na rede neural. Foi escolhida a primeira entrada como o parâmetro A, a segunda como o parâmetro B e a terceira entrada como a variável x (que não deve ser confundida com a entrada tridimensional $\mathbf{x}$), sendo tal escolha de ordem a princípio é arbitrária, porém deve estar de acordo com o domínio da função a ser definido a seguir. A função ‘ **dde.grad.jacobian()**’ é empregada para calcular derivadas de primeira ordem, de forma que a componente i=0 (implícita, por ser a padrão) da saída da rede neural ‘y’ é derivada com respeito a componente j=2 da entrada, que representa a variável x; e ‘**dde.grad.hessian()**’ é usada para calcular a derivadas da componente 0 da saída (implícito, também por ser padrão), representada

por y, em relação s componentes da entrada i=2, e j=2, ou seja, uma derivada dupla em relação a componente x. É argumentado por [25] que estas funções são mais eficientes do que as usuais "**tf.gradients()**" usadas em [55, 56, 1], por se tratar de um algoritmo "preguiçoso", onde resultados inalterados não são calculados, otimizando o processo de diferenciação.

Deve ser frisado que a função **pde**, representa a função resíduo definida em (2.5), que será avaliada nos "pontos de colocação" dentro do domínio conforme (2.8) dependendo apenas de pontos dentro do domínio e não de solução da equação diferencial. Esta representa a função custo responsável por informar à máquina a equação diferencial que está sendo estudada, e, uma vez que esta função for minimizada, é esperado que a rede neural tenha sido treinada para representar a equação diferencial, não somente nos pontos treinados, mas também no domínio da função em que foi treinada.

O domínio da função e o intervalo dos parâmetros são definidos a seguir:

```
time = dde.geometry.TimeDomain(0, 1.58)
geom = dde.geometry.Rectangle([0,0],[1,1])
geomtime = dde.geometry.GeometryXTime(geom,time)
```

Nota-se que foi utilizado por comodidade um domínio temporal chamado *time* utilizando a função "**TimeDomain**", de forma que $\{t \in [0,1.58]\}$, sendo este utilizado para mapear a coordenada **x** da equação (3.1). Também foi definido um domínio geométrico representado por um retângulo composto por um vértice inferior esquerdo na origem (0,0) e um vértice superior direito na coordenada (1,1). Tal superfície retangular é representada em coordenadas cartesianas pelo conjunto $\{(x,y) \mid x \in [0,1]\,,\, y \in [0,1]\}$, sendo este conjunto utilizado para representar as variáveis A e B. Por fim "geomtime" é definido como a união dos conjuntos "geom" e "time", representando um conjunto de dados tridimensional. Em suma, foi utilizada a representação:

$$\begin{aligned} t &\mapsto x, \\ x &\mapsto A, \\ y &\mapsto B, \end{aligned} \tag{3.4}$$

de forma que

$$\{(x,A,B) \mid x \in [0,1.58], A \in [0,1], B \in [0,1]\}, \tag{3.5}$$

representa o domínio tridimensional que contém a variável x, e os parâmetros A e B, nos seus respectivos intervalos, de acordo com (3.2).

Posteriormente, as condições iniciais do problema foram especificadas:

```
ic_1 = dde.icbc.IC(geomtime, lambda x:1,lambda _, on_initial:on_initial)

def boundary(x,_):
    return np.isclose(x[2],0)

ic_2 = dde.icbc.OperatorBC(
geomtime,
lambda x, y, _: dde.grad.jacobian(y, x, i=0, j=2) - 0,
boundary
)
```

A variável "ic_1" representa a condição inicial da função e "ic_2" representa a condição inicial da derivada da função, ambas avaliadas em $x = 0$. Ambas as condições são informadas à máquina na forma de função custo, definidas da forma (2.7). De forma detalhada, **"ic_1"** possui o argumento *geomtime*, representando o domínio da função onde os dados serão gerados. Em seguida, a condição que será imposta, representada por **"lambda x: 1"**, que no *python* significa uma função anônima que tem como entrada "x", e tem como saída o valor 1; tal função deve ser aplicada somente nos pontos em que a condição inicial é satisfeita, o que é feito pelo próximo argumento. O argumento **"lambda _, on_initial:on_initial"** é utilizado, fazendo uso da função interna do DeepXDE **"on_initial"** que checa se o ponto em questão está no início do domínio definido pela função **TimeDomain**, caso verdadeiro, a função anteriormente definida é aplicada, uma vez que esta representa a variável x, e o início do domínio é o valor 0, garante a condição $y(0) = 1$.

De forma semelhante, a condição inicial para a derivada da função é imposta utilizando a função "boundary", que verifica manualmente se um ponto pertence à fronteira inicial do domínio. Essa função realiza a verificação ponto a ponto, conferindo se a segunda coordenada da entrada (correspondente à variável x) é igual a zero dentro de uma pequena tolerância. Dessa maneira, a condição inicial $y'(0) = 0$ é corretamente aplicada e incorporada ao modelo. Desta forma, ambas as condições iniciais da equação (3.3) são informadas à máquina.

Quanto à arquitetura da rede, foi escolhida uma rede sequencial totalmente conectada com uma entrada de 3 parâmetros A, B e x, quatro camadas ocultas de 200 neurônios cada e uma camada de saída com apenas um neurônio, representando $f(A, B, x)$. Para os dados, foram utilizados 2048 pontos dentro do domínio da função para calcular a derivada de acordo com a função residual e 64 pontos foram usados para as condições iniciais.

Em relação ao número de camadas ocultas e de neurônios, a escolha foi feita empiricamente, com base em trabalhos anteriores [55, 1]. Optou-se por uma configuração

considerada robusta, com mais neurônios do que os exemplos apresentados nos artigos, e com um número de camadas semelhante ao de alguns dos casos estudados.

As configurações escolhidas incluem as seguintes: inicializador **"Glorot uniform"**, função de ativação **"Tanh"** e otimizador **"Adam"**. Também, foi adicionado uma solução do problema indicada por **"solution = func"**, assim como a métrica utilizada foi o erro L2 relativo, relacionados à avaliação dos erros. A taxa de aprendizado **lr** foi fixada em 1/1000.

```
data = dde.data.TimePDE(geomtime, pde, [ic_1, ic_2], num_domain = 2048,
num_boundary = 0, num_initial=64, solution=func, num_test=None)

layer_size = [3] + [200] * 4 + [1]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.nn.FNN(layer_size, activation, initializer)
model = dde.Model(data, net)
model.compile("adam", lr=0.001, metrics=["l2 relative error"])
```

O método representa um algoritmo *meshfree*, em que um *grid* não é especificado, porém os pontos no domínio podem ser interpretados como um análogo, representando uma resolução informada à máquina do domínio, e que será usado para calcular a equação diferencial que representa o problema em questão. Já os pontos nas condições iniciais, são dados que realmente dependem de um conhecimento prévio da equação diferencial, ou ainda, de imposições para a resolução. Desta forma, algo a ser destacado é a possibilidade de utilização do método com apenas 64 pontos da "solução" da equação diferencial no início do domínio, assim como os 64 pontos de conhecimento, ou imposição, para a derivada de primeira ordem no início do domínio.

Mesmo considerando os 2048 pontos no domínio, e no total 2176 pontos (contando com as 3 entradas, 6,528 dados), é memorável o fato de uma rede neural se propor a resolver um problema com uma quantidade desta magnitude de dados, muito pequena, comparada com redes neurais usuais. Por exemplo, o conjunto de dados de treinamento MNIST é composto por 60.000 imagens de dígitos manuscritos, enquanto outras 10.000 imagens são reservadas para teste. Cada imagem possui resolução de 28×28 pixels, totalizando aproximadamente 47 milhões de valores de pixel em todo o conjunto de treinamento. Tal feito só é possível uma vez que as funções customizadas conseguem agregar conhecimento prévio ao problema, permitindo uma limitação do espaço de busca de generalização da rede neural para soluções que representem um problema físico e também que respeitem as condições iniciais impostas.

Com o modelo definido e compilado basta treinar o mesmo pelo número de épocas escolhido, representado pela linha de código "**model.train()**" logo em seguida, podendo ser verificado no código completo disponibilizado no *github*. A evolução do treinamento da função custo será apresentada na seção de resultados.

3.1.3 Validação dos resultados

A partir do modelo treinado, é necessário avaliar sua proximidade com a solução real. Como se trata de uma equação diferencial ordinária, a solução pode ser encontrada facilmente para todo o domínio, por exemplo, utilizando o software Mathematica (Figura 7).

Partindo da solução presente na Figura 7, esta foi implementada para conferência, notando-se que a solução admite números complexos, mas produz resultados sempre reais. E também, tratando o caso especial de indeterminação da forma divisão por zero, uma vez que se trata de um limite fundamental. A implementação da solução está representada a seguir:

```
def func(x):
A = np.complex64(x[:,0:1])
B = np.complex64(x[:,1:2])
t = np.complex64(x[:,2:3])

result=np.zeros([np.shape(x)[0],1],dtype=complex)
for i in range(np.shape(x)[0]):
    if(A[i]**2 !=4*B[i]):
        result[i] = np.exp(-A[i]*t[i]/2)*(np.cosh(1/2*np.sqrt(A[i]**2\
        -4*B[i])*t[i])+(A[i]*np.sinh(1/2*np.sqrt(A[i]**2-4*B[i])\
        *t[i]))/np.sqrt(A[i]**2-4*B[i]))
    else:
```

Figura 7 – Solução equação diferencial de segunda ordem com coeficientes variáveis

```
In[2]:= $Assumptions = Element[A, Reals] && Element[B, Reals] && Element[x, Reals]
Out[2]= A ∈ ℝ && B ∈ ℝ && x ∈ ℝ

In[3]:= FullSimplify[
          DSolve[{y''[x] + A * y'[x] + B *y[x] == 0, y[0] == 1, y'[0] == 0}, y[x], x]]
Out[3]= {{y[x] → e^(-A x/2) (Cosh[1/2 √(A^2 - 4 B) x] + A Sinh[1/2 √(A^2 - 4 B) x] / √(A^2 - 4 B))}}
```

Fonte: Elaborado pelo autor.

```
            result[i] = np.exp(-A[i]*t[i]/2)*(np.cosh(1/2*np.sqrt(A[i]**2\
            -4*B[i])*t[i])+A[i]*t[i]/2)

    return np.real(result)
```

Em relação ao modelo, uma vez que tenha sido treinado, sua utilização segue exatamente como a de uma função. Por exemplo, para gerar uma predição com três entradas, o análogo de $f(A,B,x)$ pode ser feito da seguinte forma:

```
model.predict(np.array([[0.1,0.1,1]]))
```

Tem como *output*:

```
array([[0.5833357]], dtype=float32)
```

Representando a solução encontrada da equação diferencial no caso em que os parâmetros são $A = 0.1$, $B = 0.1$ e a coordenada $x = 1$, destaca-se que a rede neural treinada apresenta uma solução contínua. Portanto, não é necessário o cuidado de gerar soluções em pontos específicos do domínio, nem realizar interpolações.

O estudo em questão aborda a fixação dos parâmetros A e B do modelo, visando comparar os resultados obtidos com a função real. A representação desses resultados será apresentada na seção de resultados.

Para este caso, assim como nos demais testes, utilizou-se a métrica de erro relativo L2 para quantificar a precisão das soluções. A definição e justificativa para a adoção dessa métrica encontram-se na Seção 3.4.

A métrica foi aplicada ao modelo como um todo, mas também foi usada uma abordagem mais específica para avaliar os erros em relação a valores específicos dos parâmetros A e B. Para isso, dado um valor fixo de A e B, os resultados foram gerados para os intervalos de x, e os erros L2 relativos médios desses resultados foram calculados. Em seguida, ao varrer os intervalos de A e B enquanto mantinha o intervalo de x fixo em $x \in [0,1.58]$, foi possível gerar um gráfico dos valores médios de erro L2 relativo em relação aos parâmetros A e B. Este resultado será apresentado no Capítulo 4.

3.2 Soluções do tipo sóliton da Equação KdV

Será apresentada a metodologia das PINNs aplicadas agora em uma equação diferencial parcial não linear, que apresenta soluções do tipo sóliton, sendo mais especificamente estas do tipo *lump*. Serão tratadas as soluções para o caso de um sóliton, assim como

para o caso de dois sólitons, uma análise sistemática de otimização de hiperparâmetros e a incorporação de leis de conservação adicionais como vínculos extras.

3.2.1 Elucidando o problema

A equação KdV faz parte de uma importante classe de equações que admitem soluções do tipo sólitons, como introduzido na Seção 2.5.1. O estudo dessa equação é crucial para compreender a natureza dos sólitons e suas interações. Muitos pesquisadores já investigaram a equação KdV (2.27), que admite soluções analíticas do tipo sóliton [5].

Além dos métodos clássicos, é possível buscar soluções da equação KdV usando o método das PINNs apresentado na seção 2.2, o que permite a comparação entre os resultados obtidos pelos dois métodos para analisar mais a fundo a equação KdV e explorar essa representação de um problema físico.

Para a solução de um sóliton, um problema de valor inicial possível para a equação KdV em uma dimensão, usando as equações (2.27) e (2.38), pode ser dado por:

$$\begin{aligned} &u_t + 6uu_x + u_{xxx} = 0, \quad x \in [-5,5], \quad t \in [-1,1], \\ &u(0,x) = \frac{1}{2}\beta \operatorname{sech}^2\left(\frac{x\sqrt{\beta}}{2}\right), \quad \beta = 4.5. \end{aligned} \tag{3.6}$$

Ao examinar a literatura, é possível definir a solução de N-solitons, com destaque para a solução analítica do caso de dois solitons [57], que é dada por:

$$u(t,x) = 12\frac{3 + 4\cosh(2x - 8t) + \cosh(4x - 64t)}{(3\cosh(x - 28t) + \cosh(3x - 36t))^2}, \tag{3.7}$$

Assim, é possível estabelecer um problema de valor inicial para o caso da solução de dois sólitons, substituindo $t = 0$ na equação (3.7).

$$\begin{aligned} &u_t + 6uu_x + u_{xxx} = 0, \quad x \in [-5,5], \quad t \in [-1,1], \\ &u(0,x) = 6\,sech^2(x) \end{aligned} \tag{3.8}$$

3.2.2 Implementação

Serão apresentadas as etapas utilizadas na implementação relacionadas aos sólitons da equação KdV.

3.2.2.1 Implementação do caso original da equação KdV para um e dois sólitons

Define-se a função $f(t,x)$, conforme especificado na seção 2.2, como:

$$f := u_t + 6uu_x + u_{xxx}. \tag{3.9}$$

A implementação dessa função é relativamente simples, uma vez que a biblioteca DeepXDE possibilita a diferenciação de redes neurais em relação às suas entradas.

```
        def pde(x, u):

            u_t = dde.grad.jacobian(u, x, i=0, j=1)
            u_x = dde.grad.jacobian(u, x, i=0, j=0)
            u_xx = dde.grad.hessian(u, x, i=0, j=0)
            u_xxx = dde.grad.jacobian(u_xx, x, i=0, j=0)


            return (
                u_t
                + 6*u*u_x
                +  u_xxx
            )
```

Como mencionado anteriormente, esse método é um processo de aprendizado movido por dados, sendo necessário um conjunto de dados para sua aplicação. Foi especificado um domínio onde os pontos utilizados no *dataset* serão gerados, correspondendo ao domínio especificado em (3.6) e (3.8), sendo este definido da seguinte forma:

```
        geom = dde.geometry.Interval(-5, 5)
        timedomain = dde.geometry.TimeDomain(-1, 1)
        geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

A implementação da função que apresenta a solução da equação KdV para um ou dois sólitons pode ser observada abaixo.

```
def func(x):
    beta = 4.5
    x, t = x[:, 0:1], x[:, 1:2]

    u_1-sol = beta*0.5*(1/np.cosh(np.sqrt(beta)*0.5*(x-beta*t)))**2

    u_2-sol = 12*(( 3+4*np.cosh(2*x - 8*t) + np.cosh( 4*x \
    - 64*t))/(3*np.cosh(x - 28*t)+np.cosh(3*x-36*t))**2)

    return u_1-sol#ou return u_2-sol
```

Sendo que `u_1-sol` representa a solução para o caso de um sóliton e `u_2-sol` a solução para o caso de dois sólitons. Ambos serão usados para conferência e validação, mas também como uma simplificação da definição da condição inicial:

```
def boundary(x,_):
    return np.isclose(x[1],0)

ic_1 = dde.icbc.IC(geomtime, func, boundary)
```

A condição inicial será aplicada diretamente a partir da solução implementada, mas somente quando a condição inicial for satisfeita, isto é, no caso em que $t = 0$. Assim, basta utilizar a solução do tipo um ou dois sólitons em cada respectivo caso.

Como a escolha da condição inicial ocorreu em $t = 0$ e o domínio está definido para t no intervalo $[-1,1]$, não havia uma maneira direta de garantir que a condição seria satisfeita para um número suficiente de pontos. Isso se deve ao fato de que, na implementação, os pontos são selecionados no domínio de forma aleatória, e a função `num_initial` no DeepXDE adiciona pontos no início do domínio temporal t, que corresponderia a $t = -1$.

Foi necessário adicionar manualmente pontos em que a coordenada temporal t correspondesse a 0. Isto foi feito da seguinte forma:

```
sampler = sp.stats.qmc.Sobol(d=1,scramble=False, seed=42)
sample = sampler.random(100)
x1 = (sample -0.5)*10#(-5,5)
t0 = np.vstack(np.zeros(100))
X_anchors=np.concatenate((x1,t0), axis=1)
```

Foi empregada a biblioteca do Scipy para gerar 100 pontos quasi-aleatórios [58], que representam sequências de baixa discrepância, implementadas por oferecerem vantagens significativas sobre números pseudoaleatórios para certos problemas numéricos, ao fornecerem pontos de amostragem mais uniformes e igualmente distribuídos até mesmo em espaços multidimensionais. Especificamente, a função **Sobol** foi utilizada para gerar números no intervalo [0,1], os quais foram então re-normalizados para o intervalo $x \in [-5, 5]$, e concatenados com a coordenada $t = 0$. Os dados resultantes foram armazenados na variável `X_anchors` e adicionados ao conjunto de dados.

```
data = dde.data.TimePDE(
    geomtime,
```

```
        pde,
        [ic_1],
        num_domain=2000,
        num_boundary=0,
        num_initial=0,
        solution=func,
        num_test=None,
        anchors=X_anchors
    )
```

O número de pontos no domínio (`num_domain`) foi definido como 2000, com 0 pontos nas fronteiras (`num_boundary`). Não foram utilizados pontos nas condições iniciais (`num_initial`) porque estes foram adicionados manualmente ao conjunto de dados (`anchors`). A solução (`solution`) foi incluída para validar os resultados. Não foram usados pontos para o caso de teste (`num_test`) devido a possíveis erros quando usado com ancoras definidas. A validação com pontos de teste pode ser realizada após a finalização do treinamento, sem perdas.

Os argumentos `pde` e `ic_1` representam funções custos da rede neural sendo definidos como:

$$MSE = MSE_u + MSE_f.$$

Em que `pde` representa MSE_f e `ic_1` representa MSE_u, e são definidos conforme as equações (2.7) e (2.8).

Quanto à arquitetura da rede, foi escolhida uma rede sequencial totalmente conectada com uma entrada de 2 parâmetros x e t, cada um relacionado a um neurônio de entrada. As quatro camadas ocultas consistem em 100 neurônios cada, utilizando o mesmo critério anterior de escolha de camadas, e há uma camada de saída com um neurônio, representando $f(x, t)$.

```
layer_size = [2] + [100] * 4 + [1]
```

As configurações escolhidas seguem o padrão do exemplo anterior, incluindo: inicializador `Glorot uniform`, função de ativação `Tanh` e otimizador `Adam`. A taxa de aprendizado do Adam foi 0.001, com decaimento inverso ao tempo.

```
activation = "tanh"
initializer = "Glorot uniform"
```

```
net = dde.nn.FNN(layer_size, activation, initializer)

model = dde.Model(data, net)
model.compile(
    "adam",
    lr=0.001,
    metrics=["l2 relative error"],
    decay=("inverse time", 3000, 0.9)
)
losshistory, train_state = model.train(iterations=50000)
```

Em que o modelo foi treinado por 50 mil épocas no caso de um sóliton e 100 mil épocas no caso de dois sólitons, os resultados serão tratado na seção resultados.

3.2.3 Otimização da arquitetura da rede neural

Uma vez que foi definida a metodologia de se criar uma rede neural informada para a física com o intuito de representar as soluções da equação KdV do tipo um ou dois sólitons, foi proposto um estudo sobre otimizações da arquitetura da rede neural utilizada.

3.2.3.1 Implementação

Os testes foram sistematizados da seguinte forma:

Foi definido um intervalo de 1 a 10 camadas ocultas da rede neural, e uma variação de 20 em 20 neurônios, compreendendo o intervalo de 20 a 120 neurônios.

Tal tarefa foi implementada em *loop*, enquanto foi fixado o número de épocas em 50 mil.

```
layers=(1,10)
neurons=(0,5)

error_list=[]
for i in range(layers[0],layers[1]+1):

    for j in range(neurons[0],neurons[1]+1):

        print(i, " Layers, ", 20+j*20, " neurons:\n")

        (...)
```

```
        layer_size = [2] + [20+j*20] * i + [1]
```

A avaliação dos resultados foi conduzida da seguinte maneira: a rede neural foi treinada sem avaliação durante as primeiras 45 mil épocas. Nas 5 mil épocas finais, a rede foi monitorada com o objetivo de minimizar flutuações nos resultados. Durante esse período, a performance da rede foi avaliada a cada mil épocas, totalizando cinco avaliações. O melhor resultado obtido nessas avaliações foi salvo para conferência posterior.

Como mencionado anteriormente, é possível realizar conferências adicionais utilizando o modelo treinado. Para isso, foram utilizados 65.536 pontos de teste (256^2), um número relativamente grande que, além de ser justificado pela rapidez na avaliação do modelo já treinado, também contribui para uma avaliação mais robusta do desempenho do modelo. É importante ressaltar que os pontos de teste foram escolhidos de forma uniforme em todo o domínio de interesse. As predições geradas pelo modelo foram comparadas com as soluções exatas disponíveis na literatura, previamente definidas, para os casos de um sóliton e dois sólitons.

```
error_indvidual=[] #é criada uma lista vazia dos erros individuais
        losshistory, train_state = model.train(iterations=45000)
        for k in range(5):
            losshistory, train_state = model.train(iterations=1000)

            x_test = geomtime.uniform_points(65536, boundary=True)
            y_true = func(x_test)
            y_pred = model.predict(x_test)
            error_indv=dde.metrics.l2_relative_error(y_true, y_pred)
            print("Erro nesta iteração: ",error_indv,"\n")
            error_indvidual.append(error_indv)

        (...)

        error= min(error_indvidual) #valor minimo dos erros individuais

        print("\nErro: ",error)
        np.append(error_list, error) #os erros são salvos para conferência
```

3.2.4 Leis de conservação

A equação KdV conta com leis de conservação, mais especificamente, infinitas leis de conservação [5], sendo as três primeiras leis de conservação dadas pelas equações (2.41),

(2.42) e (2.43) definidas por:

$$\begin{aligned}
&(u)_t + \left(3u^2 + u_{xx}\right)_x = 0 \\
&\left(u^2\right)_t + \left(4u^3 + 2uu_{xx} - u_x^2\right)_x = 0, \\
&\left(u^3 - \frac{1}{2}u_x^2\right)_t + \left(\frac{9}{2}u^4 + 3u^2u_{xx} - 6uu_x^2 - u_xu_{xxx} + \frac{1}{2}u_{xx}^2\right)_x = 0.
\end{aligned} \tag{3.10}$$

A primeira lei de conservação é uma consequência direta da equação KdV, matematicamente sendo equivalente a equação diferencial implementada anteriormente. Com o intuito de se testar vínculos extras para a equação KdV, testes foram feitos utilizando a segunda lei de conservação como um vínculo extra, tanto para o caso de um sóliton, quanto para o caso de dois sólitons.

3.2.4.1 Implementação

A implementação do caso foi realizada de maneira direta, empregando a biblioteca DeepXDE para a resolução do problema. Os vínculos foram definidos na forma de funções residuais, representadas pela equação (2.5), e incorporados ao sistema através de uma função custo. O critério de otimização adotado foi o erro quadrático médio, conforme expresso pela equação (2.8), e sua aplicação seguiu uma lista de vínculos, da seguinte forma:

```
def pde(x, u):

    u_t = dde.grad.jacobian(u, x, i=0, j=1)
    u_tt = dde.grad.hessian(u, x, i=1, j=1)

    u_x = dde.grad.jacobian(u, x, i=0, j=0)
    u_xx = dde.grad.hessian(u, x, i=0, j=0)
    u_xxx = dde.grad.jacobian(u_xx, x, i=0, j=0)
    u_xxxx = dde.grad.hessian(u_xx, x, i=0, j=0)

    u_xt=dde.grad.hessian(u, x, i=0, j=1)

    #equação KdV
    f_u=(
        u_t
        + 6*u*u_x
        +  u_xxx
```

```
)

#Segunda Lei de conservação
f_c2=(
    2*u*u_t
    + 12*u**2*u_x
    + 2*u*u_xxx
)

#Terceira Lei de conservação
f_c3=(
    3*u**2*u_t
    + 18*u**3*u_x
    - 6*u_x**3
    - u_x*u_xt
    - 6*u*u_x*u_xx
    + 3*u**2*u_xxx
    - u_x*u_xxxx

)

#return [f_u, f_c2, f_c3]
return [f_u, f_c2]
```

Foram definidas as derivadas necessárias e a lei de conservação foi explicitada em termos de derivadas temporais e espaciais da equação, sendo composta na função que representa a segunda lei de conservação `f_c2`.

Além disso, a terceira lei de conservação também foi definida, porém testes sistemáticos ainda não foram realizados em relação a essa terceira lei. O código de implementação, no qual as três leis de conservação seriam utilizadas, foi representado para elucidar o problema e expressar essa possibilidade.

3.3 Generalização da Equação de Sine-Gordon

Nesta seção será utilizado o princípio de generalização de redes neurais para equações diferenciais com coeficientes variáveis apresentado na Seção 3.1, porém num contexto de equações diferenciais parciais não lineares.

Aproveitando da capacidade de generalização das redes neurais [26] foi proposto a

generalização da equação de Sine-Gordon com coeficientes variáveis.

Uma vez que a equação de Sine-Gordon em 1+1 dimensões pode ser expressa, de uma forma mais geral por (2.52)

$$\varphi_{tt} - \varphi_{xx} + m^2 \sin \varphi = 0,$$

em que m representa o parâmetro de massa, usualmente fixo. A equação (2.52) admite solução do tipo 1-sóliton (2.53) da forma

$$\varphi_{\text{soliton}}\,(x,t) := 4 \arctan\left(e^{m\gamma(x-vt)+\delta}\right),$$

onde v representa a velocidade de propagação da onda, δ representa a fase da onda, x e t representam respectivamente as coordenadas espacial e temporal. O termo γ na equação (2.50) é dado por

$$\gamma^2 = \frac{1}{1-v^2}. \tag{3.11}$$

Foi proposta uma abordagem para o problema de EDP não linear parametrizada com duas variáveis, em que o domínio da própria EDP permanece bidimensional (espaço-temporal, com x e t). No entanto, ao incluir um parâmetro m como uma entrada adicional, a rede neural estende o problema para um espaço de entrada tridimensional (x, m, t). Nesse modelo expandido, as condições de contorno são interpretadas como planos formados ao fixar a variável espacial x ou a variável temporal t em seus limites. Isso resulta em planos nos subespaços (m, t) ou (m, x). Os pontos de dados conhecidos da solução são especificados ao longo desses planos, combinando as variações de m com os contornos fixos de x ou t.

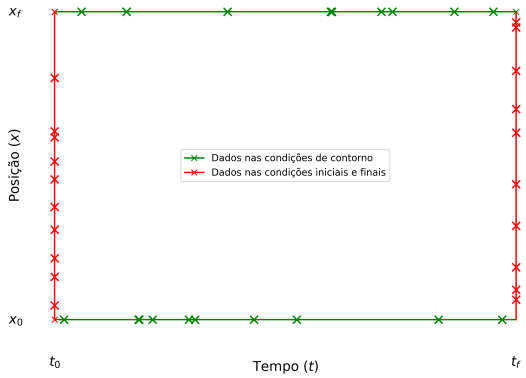
Adicionalmente, na metodologia proposta, os pontos de colocação são selecionados dentro desse espaço de entrada tridimensional expandido, denotado pelas coordenadas (x, m, t). Esses pontos são utilizados para avaliar a equação diferencial em todo o domínio espaço-temporal, enquanto incorporam as variações do parâmetro m. Apesar dos dados iniciais esparsos e dos pontos de colocação discretos, a PINN mais geral treinada busca fornecer uma solução contínua ao longo de todo o domínio expandido. Isso resulta em uma solução em 4D, na qual a solução da EDP parametrizada é projetada como uma superfície 3D para cada valor de m, produzindo, assim, uma solução generalizada para o intervalo das equações diferenciais parametrizadas.

Essa estratégia complementa a proposta inicial de utilização do método PINN para resolver a equação (2.50) de forma mais geral, permitindo a variação do termo m. O destaque do método reside em empregar um único modelo, movido por redes neurais, capaz de resolver a equação (2.52) não apenas para um valor específico de m, mas para todo um intervalo $m_0 \leq m \leq m_f$, onde os valores inicial m_0 e final m_f são determinados conforme o problema estudado.

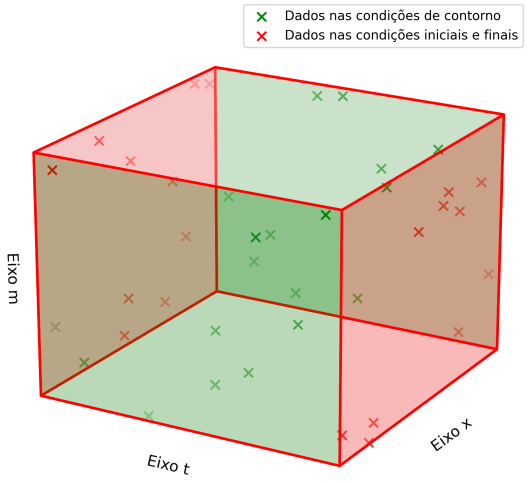
Caso bem-sucedido, esse modelo representaria uma generalização da solução, possibilitando que, uma vez treinado, bastasse fornecer um valor diferente de m para que a solução correspondente da equação diferencial fosse gerada automaticamente.

As diferenças entre o método padrão e o método proposto são ilustradas na Figura 8. No método PINN padrão, mostrado na Figura 8a, a saída da rede neural representa uma superfície 3D para um conjunto fixo de parâmetros. Em contrapartida, na abordagem modificada da PINN mais geral, ilustrada na Figura 8b, é incorporada uma dimensão adicional referente ao parâmetro, gerando um espaço de solução em 4D. Esta hipersuperfície pode ser projetada para valores fixos de m, proporcionando uma família generalizada de soluções correspondentes a diferentes valores do parâmetro.

Figura 8 – Comparação entre os métodos PINN padrão e PINN mais geral. O último incorpora uma dimensão extra de entrada, expandindo o espaço de solução para generalizar sobre uma faixa de valores de parâmetro.



(a) Método PINN padrão para resolver EDPs com duas variáveis (x e t). A rede é treinada com condições de contorno fixas para um único valor de parâmetro.



(b) Método PINN mais geral, incluindo uma dimensão adicional (m) para variação do parâmetro em um intervalo.

3.3.1 Arquitetura da rede neural utilizada

Em relação à arquitetura da rede neural, foi utilizado o padrão para as PINNs [1], ou seja, uma *feedforward neural network* (FNN), que no caso são arquiteturas relativamente simples, com função de ativação tangente hiperbólica nas camadas ocultas, e sem regularização adicional (e.g. *dropout*, etc.).

Diferentemente dos artigos originais da PINNs [55, 56], citados anteriormente, foi proposta uma ideia diferente para a camada de entrada da rede. Visando a generalização para intervalos de m, fez-se necessário que a rede neural admitisse valores variados de m, assim como valores diferentes de x e t, ou seja, além das coordenadas espaciais e tempo-

Figura 9 – *Feedforward neural network* com duas entradas.

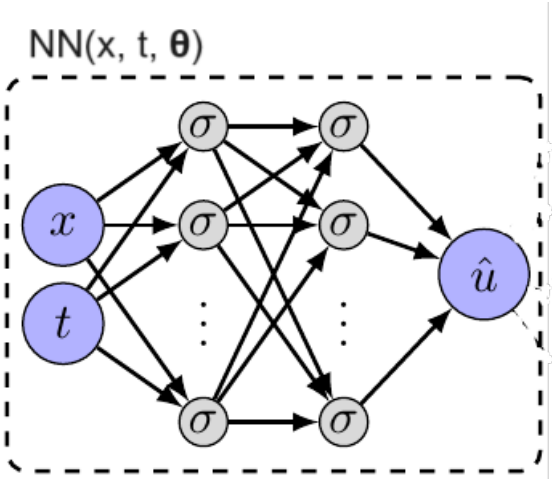
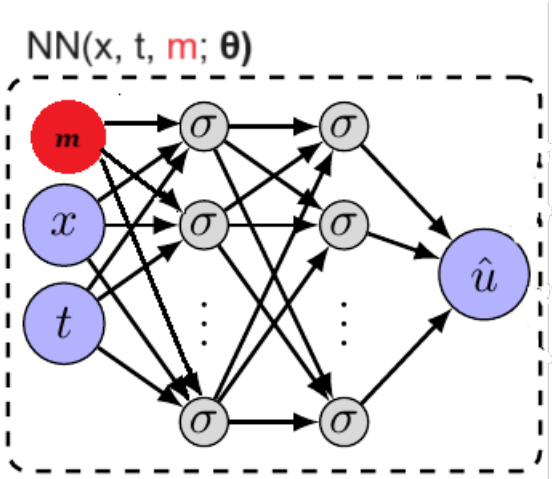


Figura 10 – *Feedforward neural network* com três entradas.



rais presentes nos artigos originais, foi proposta uma entrada adicional representando o parâmetro m.

Foi proposta uma rede com 3 entradas, primeira entrada representa a coordenada espacial x, a segunda entrada representa o parâmetro m e a terceira entrada representa a coordenada temporal t. Tal ordem foi escolhida simplesmente por simplicidade de implementação, podendo ser escolhida de forma arbitrária a princípio. Uma ilustração da rede neural utilizada originalmente pode ser representada pela Figura 9, e o comparativo com a rede neural utilizada com uma entrada extra na Figura 10, onde θ representa os parâmetros a serem treinados da rede.

Em relação às camadas ocultas, a escolha de uma arquitetura única, composta por seis camadas ocultas densas com 200 neurônios e ativação tangente hiperbólica (tanh), visa proporcionar um equilíbrio entre expressividade e generalização, permitindo também uma comparação direta entre o caso de benchmark e o cenário de teste generalizado. Embora não tenha sido realizada uma otimização arquitetural específica para cada caso, a configuração adotada foi inspirada em práticas consolidadas [55, 1], ampliando a capacidade representacional com relação ao número de neurônios. Tal abordagem evita o viés introduzido pela personalização excessiva da arquitetura para contextos específicos e reduz o custo computacional associado à busca por configurações distintas. No entanto, reconhece-se que uma análise posterior, com otimização dedicada para cada cenário, poderá revelar ganhos adicionais de desempenho — o que se propõe como uma extensão natural deste trabalho.

Pode ser observado que a rede no geral adota a característica de uma função: partindo de três parâmetros de entrada, gera um valor de saída. Porém, trata-se de uma função mais geral relacionada à equação de Sine-Gordon em 1+1 dimensão, pois possui

dependência além das coordenadas espaciais e temporais, do parâmetro de massa m. Sendo assim, faz-se necessário criar um conjunto de dados compatível com estas entradas.

3.3.2 Função resíduo, condições iniciais e de fronteira

A física previamente conhecida do problema é informada à rede neural via *loss function* (função custo). Sendo informadas via função resíduo, que representa a equação diferencial, e via condições de contorno e de fronteira conhecidas.

A equação diferencial (2.52) poderá ser escrita na forma de função resíduo (2.5) a ser informada para a máquina, utilizando $\mathbf{u}$ como notação para a solução a ser encontrada via rede neural, sendo necessária a definição dos domínios espaciais e temporais, além do intervalo de variação de m:

$$f_u := u_{tt} - u_{xx} + m^2 \sin u, \quad m \in [m_0, m_f],\ x \in [x_0, x_f],\ t \in [0, t_f]. \tag{3.12}$$

Uma vez que uma entrada extra é utilizada, esta mudança deve ser acompanhada nas definições de condições iniciais e de fronteira. Desta forma, a condição inicial será informada não somente para um valor de m fixo, mas sim para um intervalo de m.

Desta forma, para a condição inicial será fornecida a partir da solução analítica da mesma (2.50), para $t_0 = 0$ e admitindo variação para o parâmetro m

$$u(x, m, 0) = 4 \arctan\left(e^{m\,\gamma\,x\,+\,\delta}\right), \quad m \in [m_0, m_f], x \in [x_0,x_f]. \tag{3.13}$$

Como condições de contorno, foram utilizadas condições de contorno de Dirichlet, no limite inferior da forma:

$$u(x = x_0, m, t) = 4 \arctan\left(e^{m\gamma(x_0 - vt)+\delta}\right), \tag{3.14}$$

e no limite superior da forma:

$$u(x = x_f, m, t) = 4 \arctan\left(e^{m\gamma(x_f - vt)+\delta}\right). \tag{3.15}$$

Os termos de fase de onda $\boldsymbol{\delta}$ e velocidade $\mathbf{v}$ (e consequentemente fator de Lorentz $\boldsymbol{\gamma}$), nas equações (3.12), (3.13), (3.14) e (3.15) são constantes a serem definidas, tomando os devidos cuidados por se tratarem de medidas físicas.

Uma vez que a equação diferencial foi especificada, juntamente com as condições iniciais e de contorno do problema, foi possível realizar a implementação do treinamento do modelo proposto, garantindo que as restrições impostas fossem corretamente incorporadas ao processo de aprendizado.

3.4 Métricas de Avaliação

Para quantificar a precisão das soluções obtidas, adotou-se o erro relativo L2 como principal métrica de avaliação. Esta métrica é definida como:

$$\text{Erro relativo } L^2 = \frac{\|u_{\text{pred}} - u_{\text{true}}\|_2}{\|u_{\text{true}}\|_2},$$

onde $\|u\|_2 = \left(\sum_{i=1}^{N} |u_i|^2\right)^{1/2}$ é a norma L2 tradicional. O erro relativo L2 mede a diferença global entre as soluções prevista e exata, normalizada pela magnitude da solução exata. Essa abordagem permite avaliar a precisão relativa do modelo de forma robusta, independentemente da escala do problema.

A escolha desta métrica se justifica por sua ampla adoção na literatura de PINNs [55, 1], além de sua sensibilidade a desvios distribuídos ao longo de todo o domínio, o que é desejável em equações diferenciais parciais. Também foi considerada a utilização de outras métricas, como o erro absoluto médio (MAE), mas optou-se por manter o erro relativo L2 como principal critério, por fornecer uma visão mais abrangente do desempenho do modelo em regiões com maior variação da solução.

Além dessa métrica global, foram computadas e analisadas diretamente as diferenças locais entre solução exata e predição ($u_{\text{true}} - u_{\text{pred}}$) em todo o domínio. Essa abordagem adicional permitiu verificar a coerência dos resultados e a distribuição espacial dos erros, assegurando que a rede neural não apenas obtivesse bons valores médios de erro, mas também produzisse soluções localmente consistentes. Tais análises foram importantes para confirmar a veracidade física e matemática das soluções obtidas.

3.5 Análise Comparativa de Metodologias para EDPs Parametrizadas

Esta seção fornece uma análise comparativa da abordagem utilizada neste trabalho e da estratégia de meta-aprendizagem para PINNs detalhada em [59]. Ambas as metodologias visam abordar eficientemente Equações Diferenciais Parciais (EDPs) parametrizadas usando Redes Neurais Informadas pela Física (PINNs), mas adotam estratégias fundamentalmente diferentes.

3.5.1 Objetivo Principal e Tratamento de Parâmetros

As duas abordagens divergem significativamente em seus objetivos primários e em como incorporam os parâmetros da EDP:

- **Metodologia proposta:** O objetivo principal é desenvolver um único modelo de rede neural capaz de resolver uma família de EDPs ao longo de uma faixa contínua de um parâmetro específico da EDP. Isso é alcançado tratando o parâmetro variável (por exemplo, m na equação de Sine-Gordon) como uma entrada adicional explícita para a rede neural, juntamente com as coordenadas espaciais (por exemplo, x) e temporais (por exemplo, t). A rede aprende um mapeamento direto $u(x, t, \text{parâmetro})$.

- **Meta-aprendizagem [59]:** Esta abordagem busca acelerar o processo de otimização (treinamento) das PINNs ao encontrar novos valores de parâmetros não vistos anteriormente. Diferentes valores de parâmetros ξ são considerados tarefas distintas. Um meta-modelo é treinado para prever bons pesos iniciais de rede $\underline{w}(\xi)$ para uma PINN, com base no parâmetro ξ. Este conjunto de pesos previsto serve então como ponto de partida para uma otimização subsequente e mais rápida para aquele valor específico do parâmetro.

3.5.2 Arquitetura da Rede, Treinamento e Predição

As implicações para o design da rede, o pipeline de treinamento e como as soluções são obtidas são marcadamente diferentes, representadas na Tabela 1.

3.5.3 Simplicidade Conceitual e Esforço de Implementação

- **Metodologia proposta:** Este método é caracterizado por sua simplicidade conceitual e implementação mais direta como uma extensão da abordagem PINN padrão. O principal ajuste envolve modificar a camada de entrada e garantir que os dados de treinamento cubram a dimensão do parâmetro. Ao se adicionar uma dimensão de entrada para o parâmetro, um impacto mínimo no tempo de treinamento para tamanhos de rede comparáveis (Observado no Capítulo 4).

- **Meta-aprendizagem:** Esta abordagem é significativamente mais complexa, envolvendo um pipeline multi-estágio (geração de dados via K treinamentos de PINN, seleção e treinamento do meta-modelo e, finalmente, adaptação para novas tarefas). Penwarden et al. também observam considerações como garantir a suavidade dos pesos entre as tarefas para uma meta-aprendizagem eficaz. O custo computacional inicial para treinar K PINNs pode ser substancial.

3.5.4 Resumo dos Pontos Fortes

- **Metodologia proposta** Oferece predição eficiente, quase instantânea, para qualquer parâmetro dentro da faixa treinada usando um único modelo. Sua simplicidade

Tabela 1 – Comparação de Aspectos Metodológicos

Aspecto	**Modelo proposto**	**Meta-aprendizagem [59]**
Estrutura da Rede	Uma única rede neural com uma camada de entrada expandida para incluir o(s) parâmetro(s) da EDP.	Múltiplas PINNs padrão (K instâncias) para geração de dados; um meta-modelo separado (por exemplo, GP, RBF) para prever pesos; uma nova PINN para cada novo parâmetro.
Processo de Treinamento	Uma fase primária de treinamento para o único modelo generalista. Os dados de treinamento (pontos de colocação, contorno, iniciais) abrangem os domínios espacial, temporal e de parâmetros estendidos.	Multi-estágio: 1. Treinar K PINNs individuais em K amostras de parâmetros ξ_j para obter os pesos $\underline{w}(\xi_j)$. 2. Treinar o meta-modelo usando esses pares $\{\xi_j, \underline{w}(\xi_j)\}$.
Predição para Novo Parâmetro	O único modelo treinado produz diretamente a solução para qualquer valor de parâmetro dentro de sua faixa aprendida por meio de uma passagem direta (sem retreinamento).	O meta-modelo prevê pesos iniciais $\underline{\hat{w}}(\xi^*)$ para o novo parâmetro ξ^*. Uma PINN inicializada com esses pesos ainda deve passar por um processo de otimização.
Dados para o Modelo	Pontos de contorno, iniciais e de colocação são definidos sobre o domínio (x,t,m), incluindo explicitamente o parâmetro m.	O meta-modelo é treinado em pares de {valores de parâmetros ξ_j, pesos de PINN treinados correspondentes $\underline{w}(\xi_j)$}. Cada uma das K PINNs usa dados (x,t) padrão.

o torna altamente acessível. Demonstrou precisão comparável às PINNs de parâmetro fixo no exemplo de Sine-Gordon.

- **Meta-aprendizagem:** Projetado para tornar o processo de obtenção de soluções para novos valores individuais de parâmetros mais rápido, fornecendo inicializações de peso superiores, reduzindo assim o tempo de otimização para etapas subsequentes de L-BFGS, como mostrado em seus resultados.

Em essência, o método proposto neste trabalho visa criar um solucionador geral para uma faixa de parâmetros, enquanto a abordagem de meta-aprendizagem visa otimizar o processo de criação de solucionadores para instâncias específicas de parâmetros.

4 Resultados e discussões

Serão apresentados os resultados dos testes sistemáticos propostos no Capítulo 3. Além disso, tais resultados serão analisados de forma a se verificar se os objetivos apresentados foram alcançados.

Particularidades da biblioteca

O DeepXDE possui um bug conhecido e reportado referente às métricas 'Train loss' e 'Test loss' quando dados de treinamento são usados como pontos específicos ('âncoras') para definir condições iniciais ou de contorno. Nestes casos, a biblioteca, que visa ser compatível com diversos *backends*, apresenta inconsistências na definição dos dados de teste, resultando no cálculo das perdas de treino e teste sobre o mesmo conjunto de dados. Contudo, essa particularidade não compromete o processo de treinamento em si, e o modelo pode ser adicionalmente validado de forma robusta por outros meios. Foram utilizados nesta seção, plotagem dos *logs* de treinamento das redes neurais utilizadas, tais plotagens servem como ilustração do processo de treinamento, porém a escolha dos pontos ótimos de aprendizado foi feita com ajuda ferramentas de monitoramento de treinamento do DeepXDE, assim como observação atenciosa dos *logs* de treinamento

4.1 Equação diferencial ordinária de segunda ordem com coeficientes variáveis

Para o caso da equação diferencial de segunda ordem com coeficientes variáveis, três testes principais foram feitos, compreendendo três intervalos dos valores dos parâmetros A e B, enquanto as condições iniciais e o intervalo de avaliação no domínio da variável x se mantiveram iguais nos três casos.

Para a equação:

$$y''(x) + A\,y'(x) + B\,y(x) = 0, \quad x \in [0, 1.58],$$

com condições iniciais:

$$y(0) = 1, \quad y'(0) = 0.$$

Os casos consistiram nos intervalos de avaliação dos parâmetros A e B:

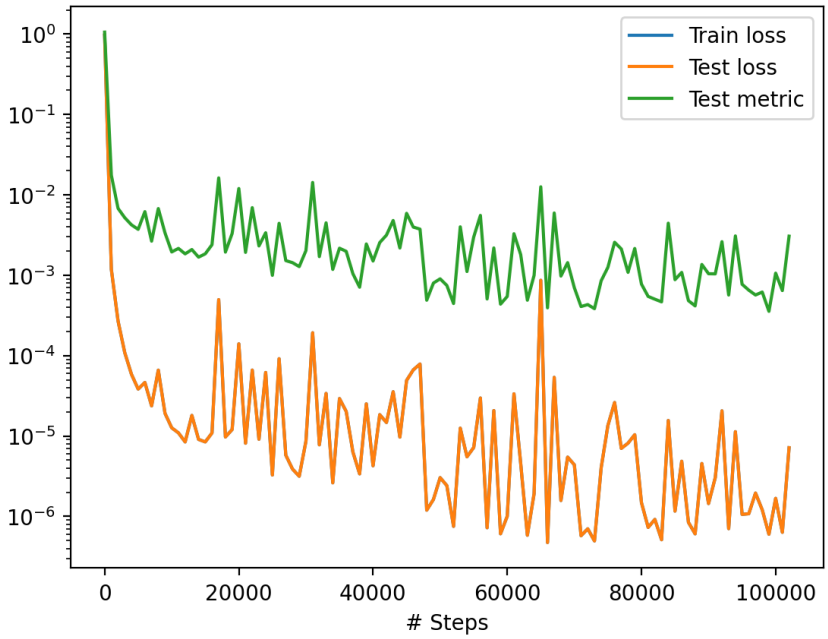
- Caso 1: $A \in [0, 1], \quad B \in [0, 1],$
- Caso 2: $A \in [-1, 1], \quad B \in [-1, 1],$

- Caso 3: $A \in [0, 4], \quad B \in [0, 4]$.

4.1.1 Caso 1

Foi empregado um algoritmo de *checkpoint*, em que os modelos com melhor otimização, avaliados de 1000 em 1000 épocas, foram salvos, sendo o máximo de épocas 100 mil. Os logs de treinamento estão disponíveis no GitHub, enquanto o gráfico da evolução dos treinamentos é apresentado na Figura 11. O valor de melhor resultado e utilizado para as avaliações compreendeu o da época 66 mil.

Figura 11 – Evolução do valor da função custo e métrica L2 durante o treinamento, caso 1: $A \in [0, 1], B \in [0, 1]$.



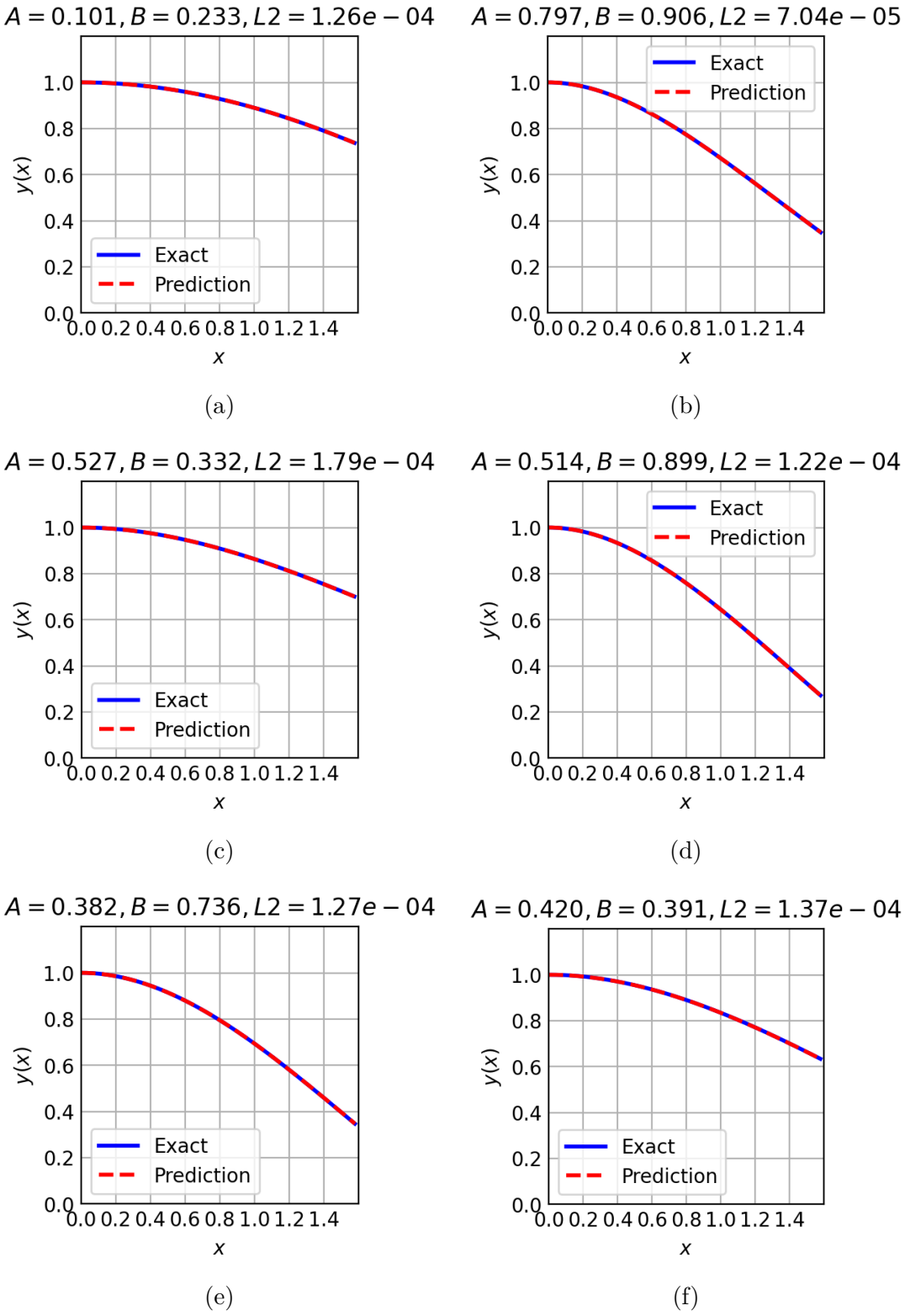
Fonte: Elaborado pelo autor.

Com o modelo treinado, este foi avaliado fixando os valores de A e B, sendo 6 exemplos de valores aleatórios para A e B apresentados na Figura 12. Também foram calculados os valores do erro L2 relativo médio para cada gráfico, sendo estes apresentados em cada subfigura.

Mesmo sendo possível avaliar de caso a caso os erros, para uma melhor visualização num contexto geral foi proposto uma forma de avaliação que percorresse os intervalos de A e B enquanto se calculava o erro L2 relativo médio de cada caso, para a função no intervalo de $x \in [0, 1.58]$, sendo estes representado na Figura 13.

No caso 1, ao se analisar o erro total do modelo, obteve-se o valor de L2 relativo médio de $6{,}260 \times 10^{-4}$.

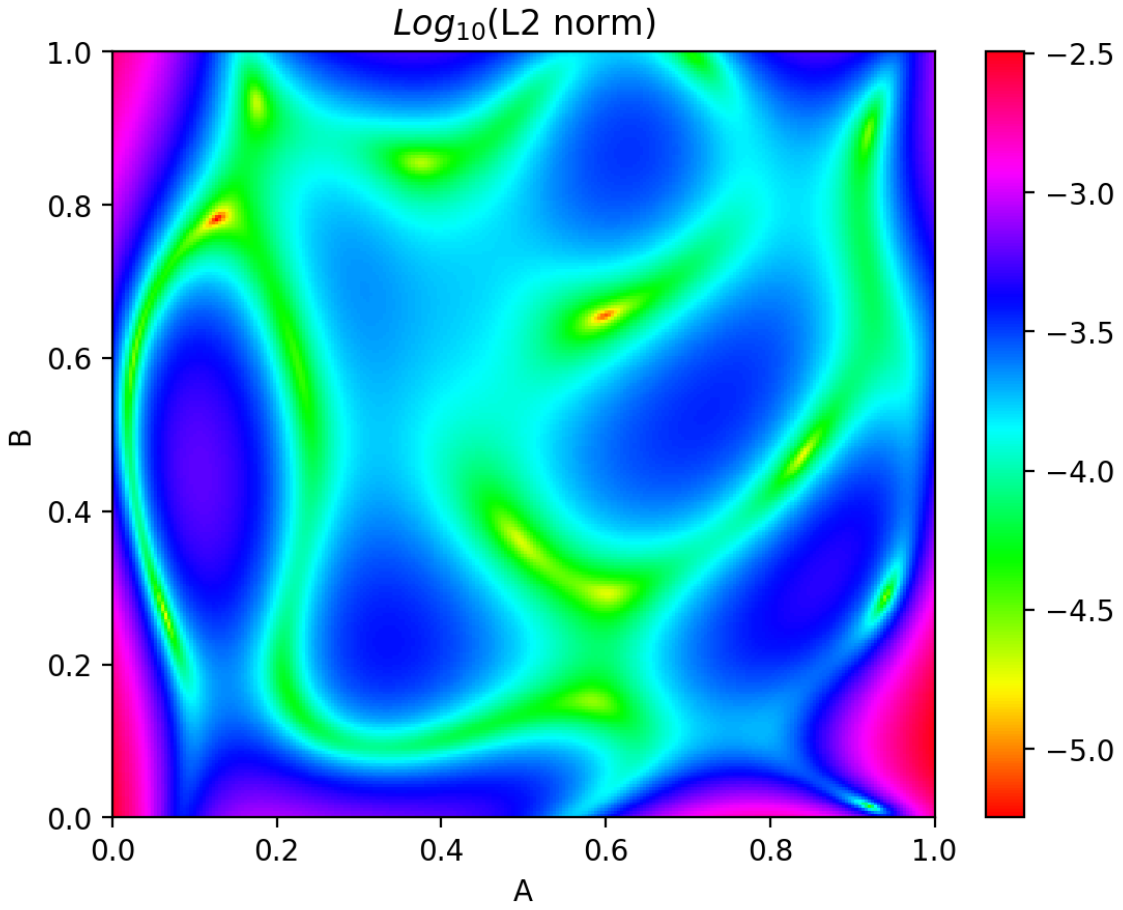
Figura 12 – Gráfico da evolução dos treinamentos no caso 1: $A \in [0,1], B \in [0,1]$.



(a) (b) (c) (d) (e) (f)

Fonte: Elaborado pelo autor.

Figura 13 – Erro L2 relativo médio para cada par de parâmetros $A \in [0, 1], B \in [0, 1]$.

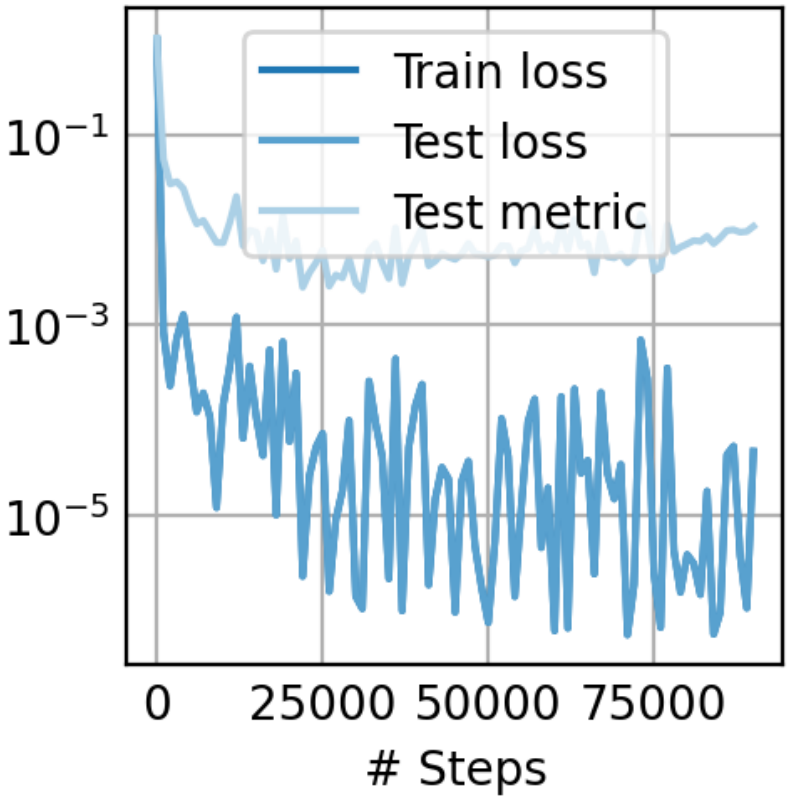


Fonte: Elaborado pelo autor.

4.1.2 Caso 2

No caso 2 apresentado previamente, empregou-se um algoritmo de *checkpoint*, em que os modelos com melhor otimização, avaliados de 1000 em 1000 épocas, foram salvos, sendo o máximo 100 mil. Os logs de treinamento estarão disponíveis no GitHub, enquanto o gráfico da evolução dos treinamentos é apresentado na Figura 14, sendo melhor resultado foi obtido na época 71 mil.

Figura 14 – Evolução do valor da função custo e métrica L2 durante o treinamento, caso 2: $A \in [-1, 1], B \in [-1, 1]$. **Obs.**: Para o *log* deste caso específico, uma interação inesperada ocorreu, desconfigurando o mesmo. Como A recuperação deste treinamento em específico seria impossível, foi mantido o gráfico, mesmo que desconfigurado.



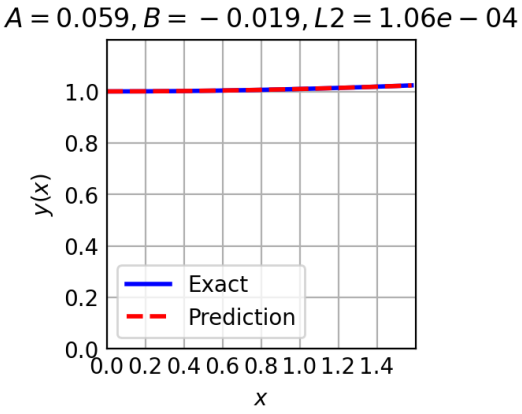
Fonte: Elaborado pelo autor.

Com o modelo treinado, ele foi avaliado fixando os valores de A e B. Seis exemplos de valores aleatórios para A e B são apresentados na Figura 15. Também foi calculado o valor do erro L2 relativo médio para cada gráfico, sendo estes apresentados em cada subfigura.

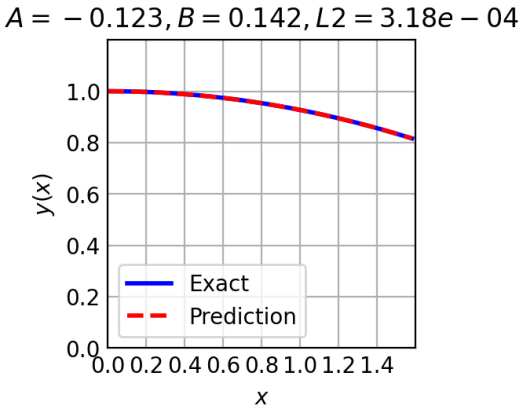
Para uma melhor visualização num contexto geral, foi proposta uma forma de avaliação que percorresse os intervalos de A e B enquanto se calculava o erro L2 relativo médio de cada caso, para a função no intervalo de $x \in [0, 1.58]$, sendo este representado na Figura 16.

Figura 15 – Gráfico da evolução dos treinamentos do caso 2: $A \in [-1, 1], B \in [-1, 1]$.

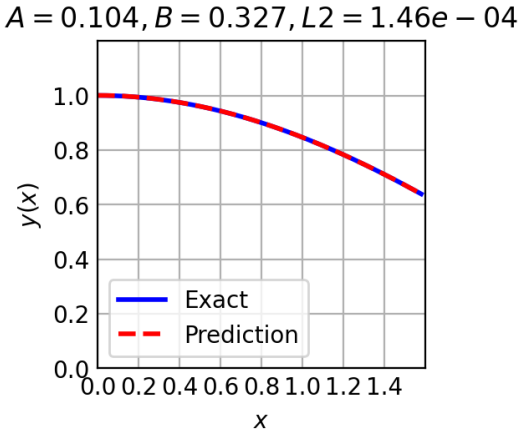
(a)



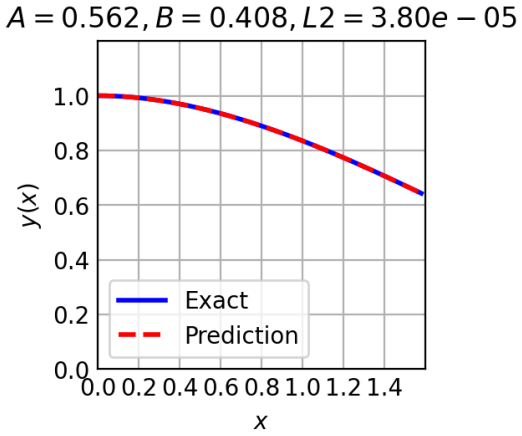
(b)



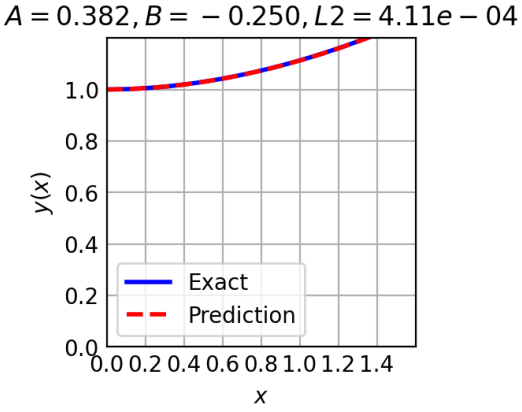
(c)



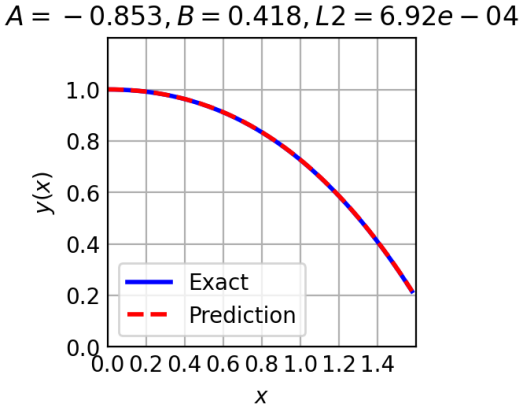
(d)



(e)

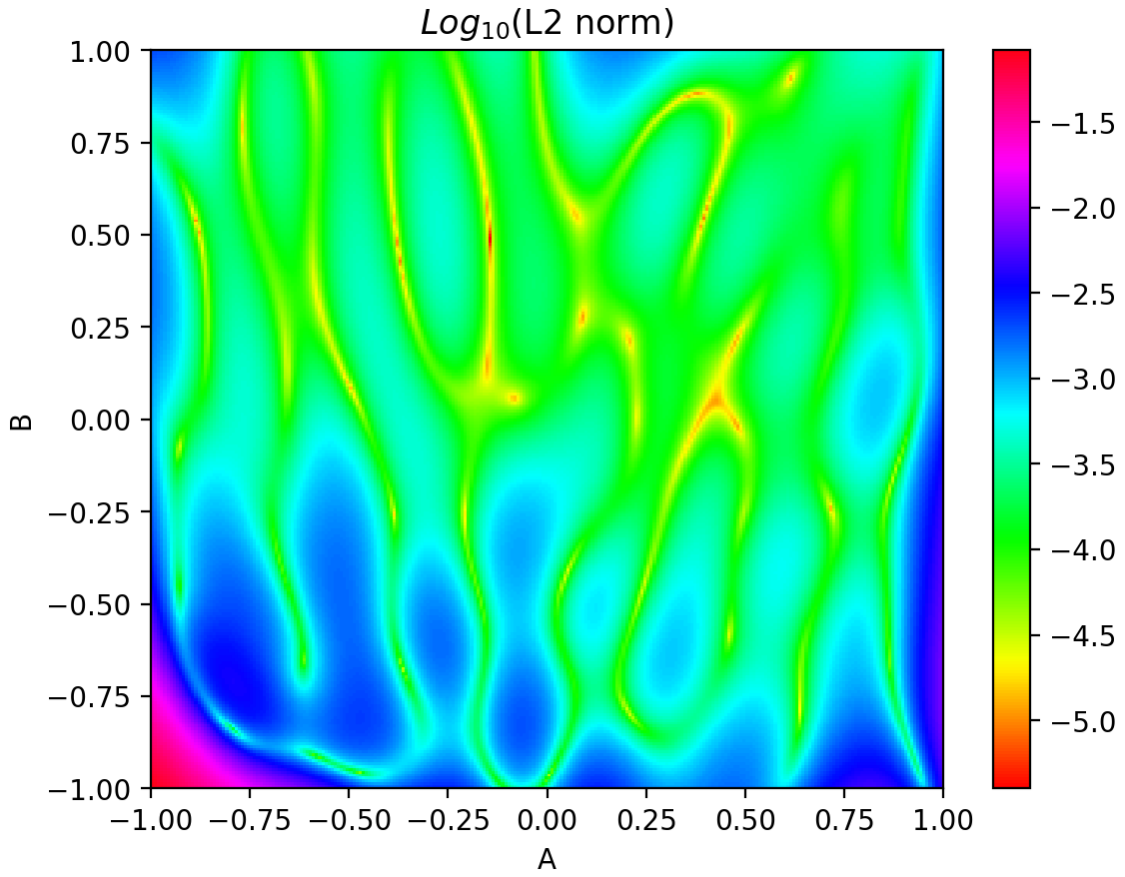


(f)



Fonte: Elaborado pelo autor.

Figura 16 – Erro L2 relativo médio para cada par de parâmetros $A \in [-1, 1], B \in [-1, 1]$.



Fonte: Elaborado pelo autor.

Analisando o erro do modelo como um todo, no caso 2 foi obtido um valor geral de L2 relativo médio no valor de $9{,}780 \times 10^{-3}$.

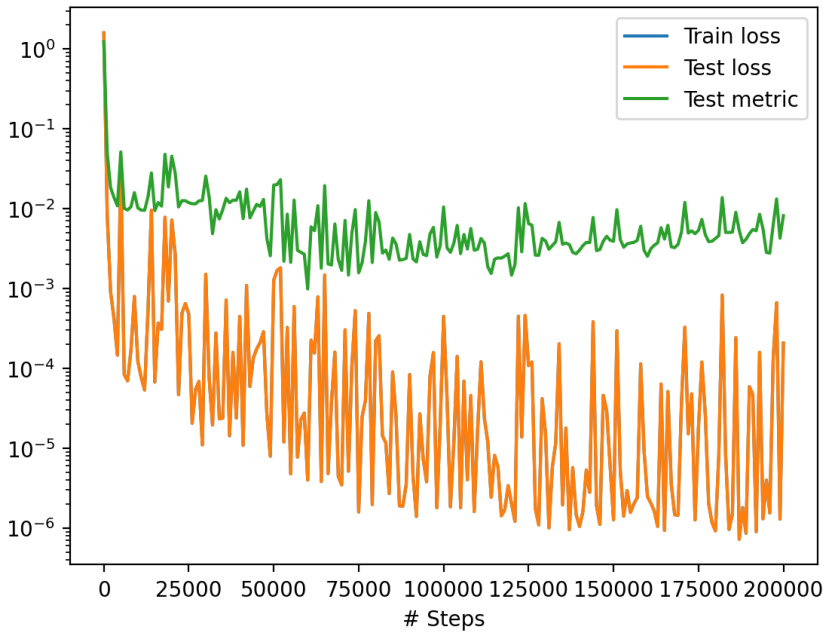
4.1.3 Caso 3

Assim como nos casos 1 e 2, também empregou-se, para o caso 3, um algoritmo de *checkpoint*, em que os modelos com melhor otimização, avaliados de 1000 em 1000 épocas, foram salvos, sendo o máximo 200 mil épocas. Os logs de treinamento estarão disponíveis no GitHub, enquanto o gráfico da evolução dos treinamentos será apresentado na Figura 17.

O melhor resultado foi obtido na época 60 mil, sendo o modelo neste estado de treinamento escolhido por apresentar a menor métrica L2. O modelo foi treinado por um total de 200 mil épocas; no entanto, o valor otimizado foi alcançado nas primeiras 100 mil épocas. A partir da análise da figura, observam-se indícios de *overfitting* após as 100 mil épocas, onde a *loss* continua a decair, mas a métrica L2 não acompanha essa redução, sugerindo uma piora no desempenho do modelo.

Com o modelo treinado, ele foi avaliado fixando os valores de A e B. Seis exemplos

Figura 17 – Evolução do valor da função custo e métrica L2 durante o treinamento, caso 3: $A \in [0, 4], B \in [0, 4]$.



Fonte: Elaborado pelo autor.

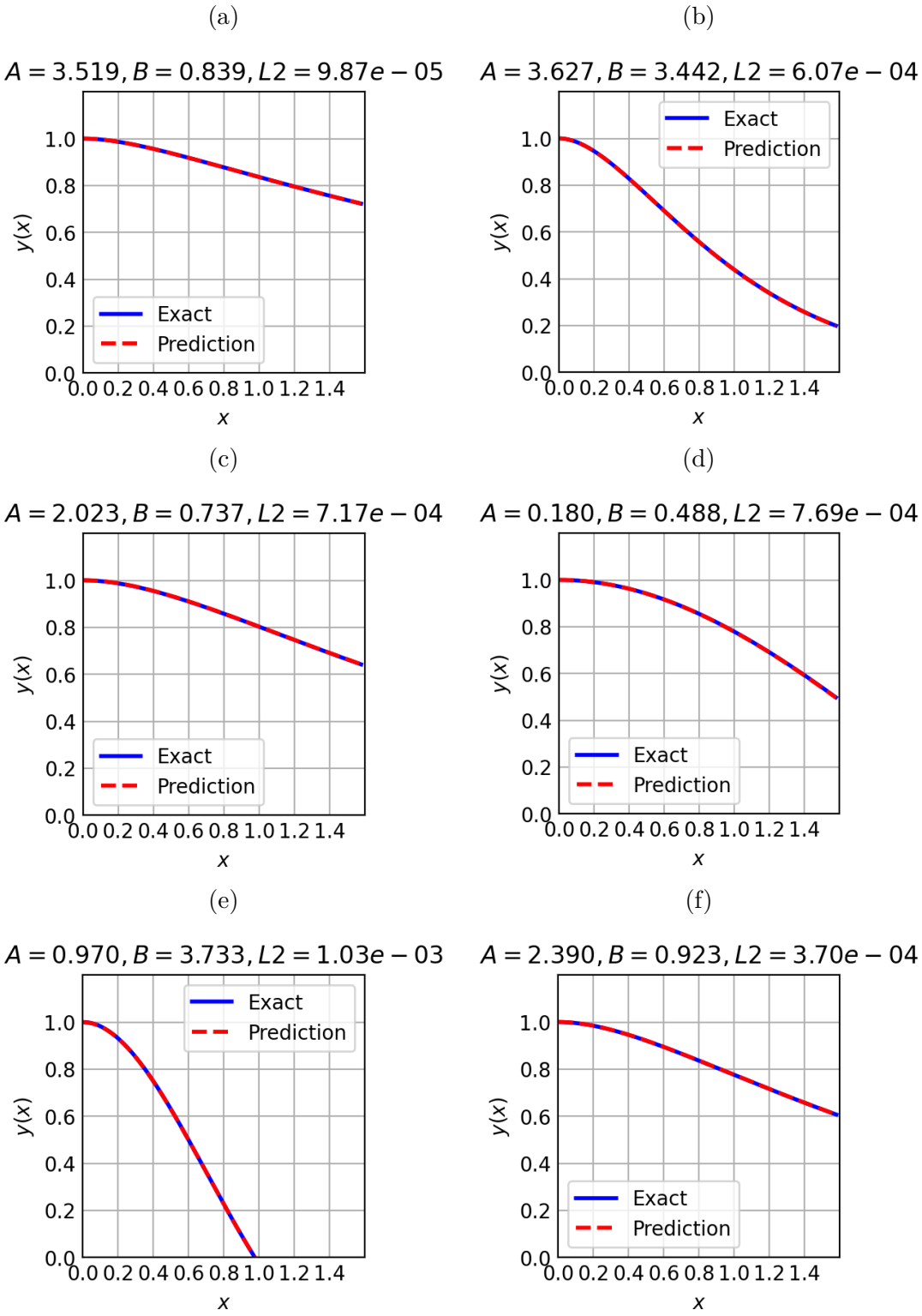
de valores aleatórios para A e B são apresentados na Figura 18. Também foi calculado o valor do erro L2 relativo médio para cada gráfico, sendo estes apresentados em cada subfigura.

Observa-se um desempenho satisfatório do modelo nos exemplos analisados, com predições consistentes em relação aos valores reais. Mesmo nos piores casos, os erros relativos permanecem da ordem de 10^{-3}, o que equivale a desvios próximos ou inferiores a 0,1%.

Para uma melhor visualização num contexto geral, foi proposta uma forma de avaliação que percorresse os intervalos de A e B enquanto se calculava o erro L2 relativo médio de cada caso, para a função no intervalo de $x \in [0, 1.58]$, sendo este representado na Figura 19.

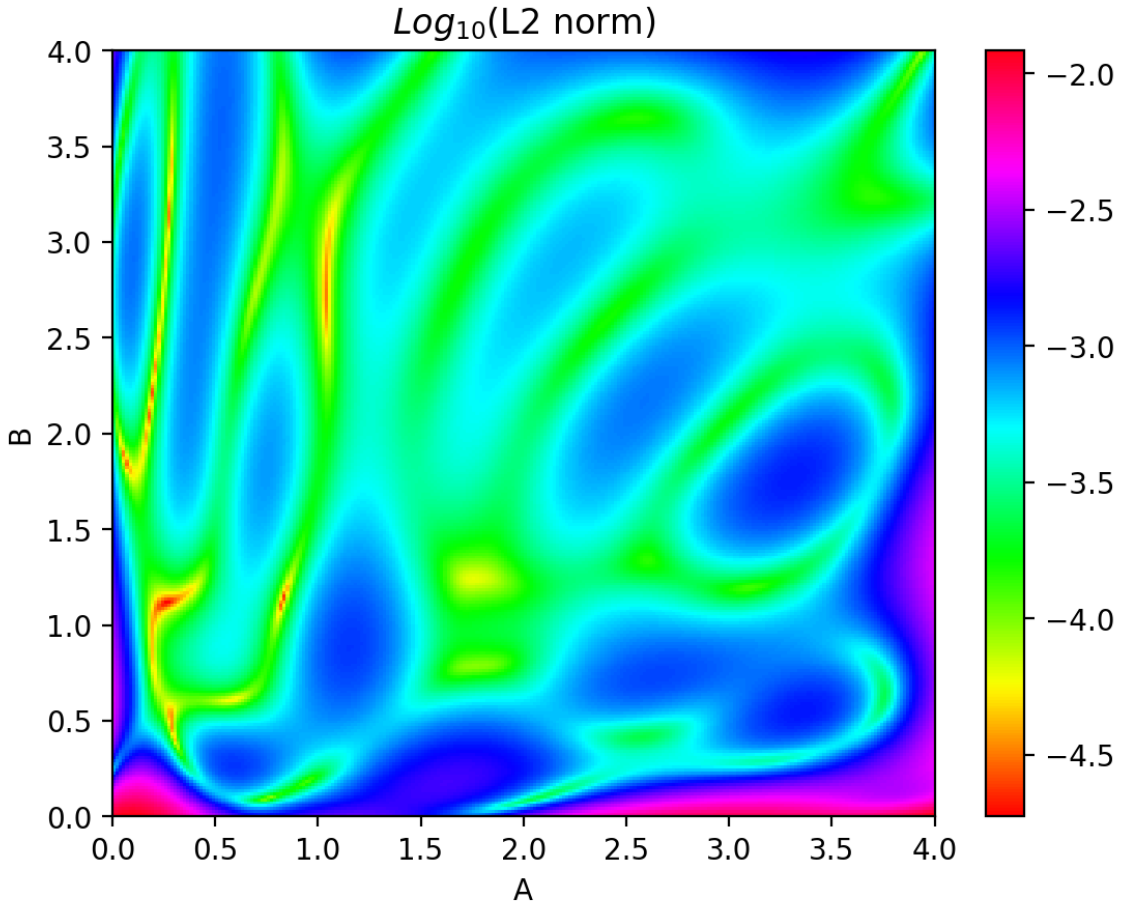
Ao analisar os resultados, obteve-se no caso 3 um valor geral de L2 relativo médio no valor de $\text{L2} = 1{,}540 \times 10^{-3}$.

Figura 18 – Gráfico da evolução dos treinamentos no caso 3: $A \in [0, 4], B \in [0, 4]$.



Fonte: Elaborado pelo autor.

Figura 19 – Erro L2 relativo médio para cada par de parâmetros $A \in [0, 4], B \in [0, 4]$.



Fonte: Elaborado pelo autor.

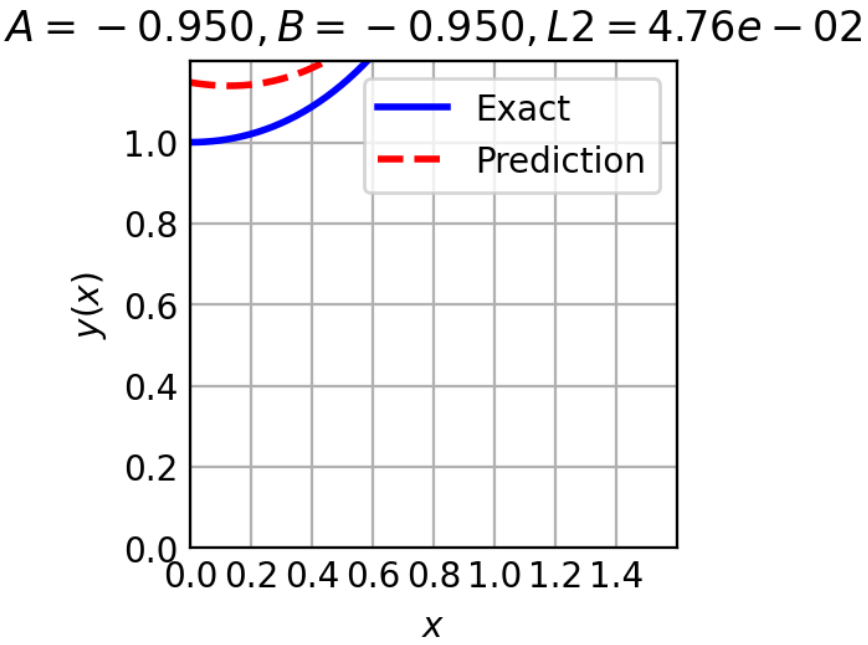
4.1.4 Discussões

Observa-se uma boa concordância entre os valores preditos pelo modelo e os valores reais. Considerando o erro relativo, o pior caso apresenta um erro médio de 0,3% no caso 2, com A e B variando de -1 a 1. Nesse cenário, há uma degradação significativa quando A e B se aproximam de -1, como mostrado na Figura 20.

Os resultados desse caso apresentam uma discrepância considerável em relação aos demais, nos quais os valores de saída, $y(x)$, geralmente pertencem ao intervalo [0; 1,2]. A forte característica exponencial positiva observada quando os parâmetros se aproximam de -1, em contraste com funções mais suaves e variações mais graduais nos outros cenários, parece ter dificultado o treinamento na região em torno de A e $B = -1$.

Em relação aos três resultados gerais (Figuras 13, 16 e 18), observou-se que os modelos apresentaram bom desempenho nas regiões centrais dos gráficos. No entanto, nas extremidades dos intervalos de A e B, as faixas vermelhas tendem a se tornar rosadas, especialmente nas partes inferior e superior dos gráficos, o que indica um aumento no erro. Esse comportamento pode ser justificado pela ausência de condições de contorno

Figura 20 – Observação sobre o pior caso de avaliação do modelo, no caso 2, com A e B se aproximando de -1.



Fonte: Elaborado pelo autor.

diretamente nessas regiões. Já nas bordas laterais, direita e esquerda, onde há condições de contorno aplicadas, é importante lembrar que essas condições foram difundidas para todos os valores de A e B do intervalo estudado. Mesmo assim, o aumento dos erros nessas regiões pode sugerir a necessidade de mais pontos de contorno para garantir maior precisão.

No geral, os erros foram relativamente baixos, cujos $\log_{10}$ dos valores foram aproximadamente -3.2 no caso 1, -2.0 no caso 2 e -2.8 no caso 3. Foi observado que mesmo o intervalo estudado no caso 3 sendo maior do que o caso 2, um resultado melhor foi encontrado. Isto indica que um cuidado na escolha de intervalos com características de soluções mais bem comportadas apresenta impacto significativo na representação de um modelo de boa generalização para funções com parâmetros variáveis. Uma possível forma de contornar esse problema seria o desenvolvimento de um modelo focado nas regiões onde ocorrem maiores discrepâncias em relação ao restante do domínio, ou ainda o uso de modelos distintos para intervalos específicos de interesse. Outra possível solução seria o uso de mais dados de condições de contorno nessas regiões de maior variação.

Foram também observados pontos específicos com erros relativos significativamente baixos, representados por transições pontuais de laranja para vermelho nos gráficos, e padrões de faixas com erros menores dentro dos intervalos de A e B, representados por faixas verdes tendendo para o amarelo. A causa desses padrões não foi estudada, sendo necessário um estudo posterior para explicar esse comportamento.

4.2 Solitons da equação KdV

Uma vez que os resultados do uso da PINN em um modelo com equações relativamente mais simples foram apresentados, serão apresentados nesta seção os resultados dos estudos da equação KdV. Diferentemente do caso anterior, o método será aplicado a uma equação diferencial parcial não linear.

Uma implementação direta do problema foi feita, em seguida foram empregados testes sistemáticos para uma análise de um modelo que melhor represente soluções do tipo 1 ou 2 sólitons da equação KdV. Finalmente, um modelo mais robusto foi desenvolvido com o uso dos resultados dos testes sistemáticos.

4.2.1 Implementação direta do problema

Afim de ter um primeiro contato com o problema, foi feita uma implementação direta, como abordado na Seção 3.2. Para este fim, foi empregada uma arquitetura semelhante à utilizada nos casos de estudo apresentados no artigo que introduziu o termo PINNs [1].

4.2.1.1 Solução de um sóliton da equação KdV via redes neurais

A fim de realizar o treinamento da rede, considere a equação diferencial e condição inicial pertinente a um sóliton, conforme a equação (3.6),

$$u_t + 6uu_x + u_{xxx} = 0, \quad x \in [-5,5], \quad t \in [-1,1],$$

$$u(0,x) = \frac{1}{2}\beta \operatorname{sech}^2\left(\frac{x\sqrt{\beta}}{2}\right), \quad \beta = 4.5,$$

sem vínculos extras.

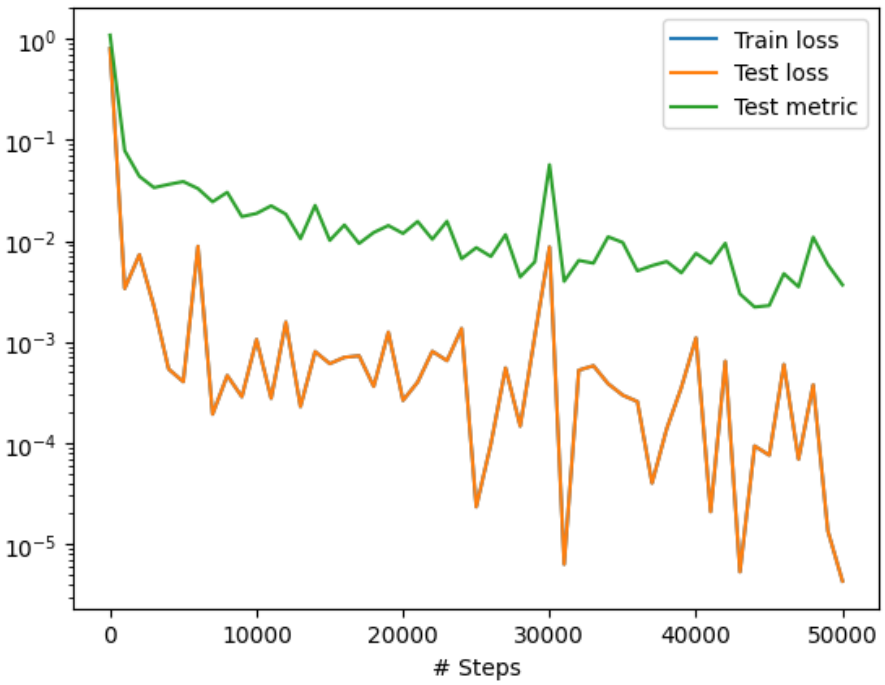
A rede neural utilizada possui uma arquitetura especificada por

```
layer_size = [2] + [100] * 4 + [1]
```

composta por uma camada de entrada com 2 neurônios, quatro camadas ocultas contendo 100 neurônios cada, e uma camada de saída com 1 neurônio. O processo de treinamento foi realizado, sendo o resultado do valor da função de custo ao longo do treinamento representado na Figura 21. A solução encontrada para a equação diferencial foi obtida, sendo representada na Figura 22.

Se observa uma concordância do resultado com a função analítica, em que o resultado encontrado possui um erro L2 relativo de 3.86×10^{-03}.

Figura 21 – Evolução do treinamento da rede neural, caso 1 sóliton

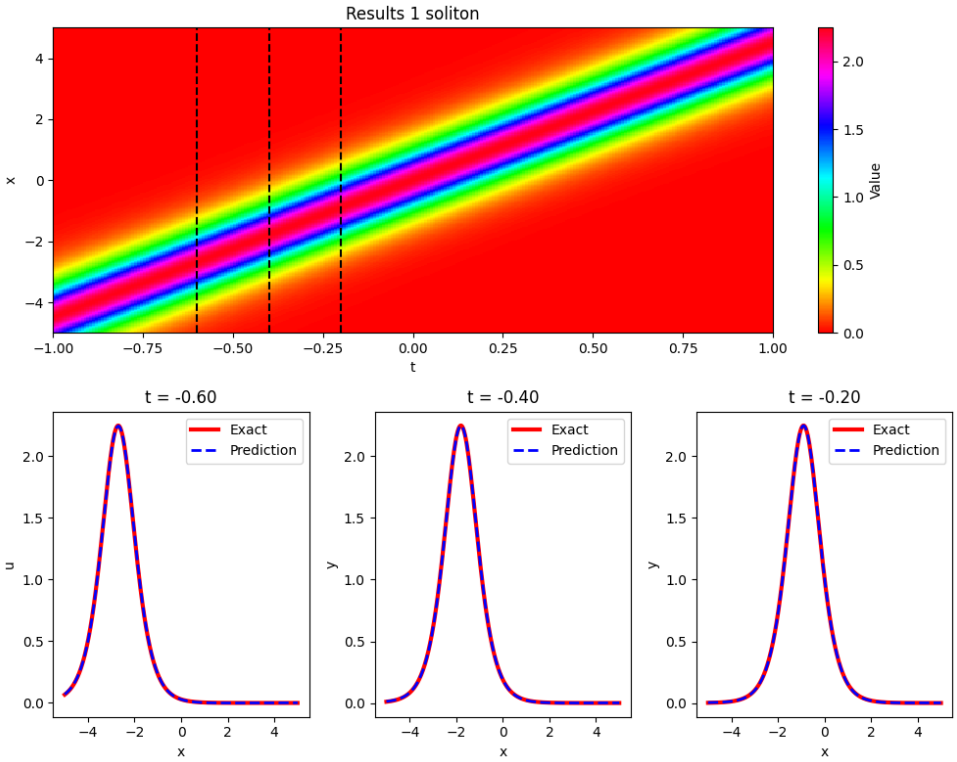


Fonte: Elaborado pelo autor.

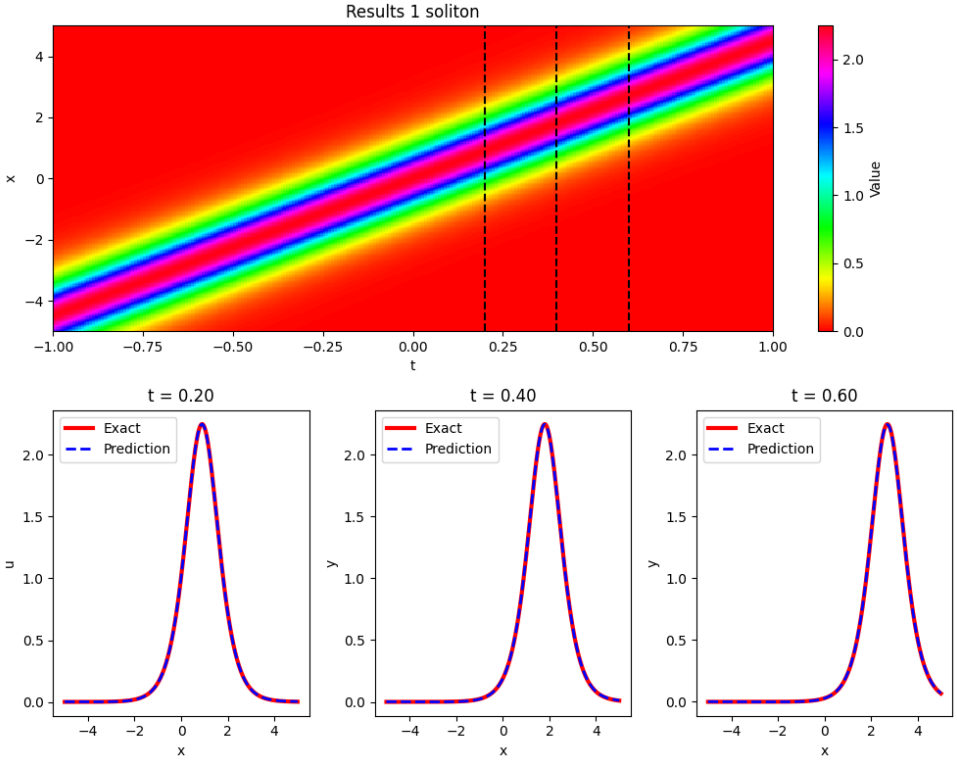
Um gráfico representando o erro, diferença do valor real e valor predito, pode ser plotado, sendo este representado na Figura 23. Pode ser observado que a condição inicial em t=0 foi respeitada pela rede neural, e conforme os resultados foram se afastando da condição inicial, foi observada uma piora nos resultados, onde, nas extremidades do domínio temporal t foi observado um erro absoluto da ordem de 10^{-2}.

Figura 22 – Resultado do treinamento de um sóliton

(a) Instantes anteriores à condição inicial.

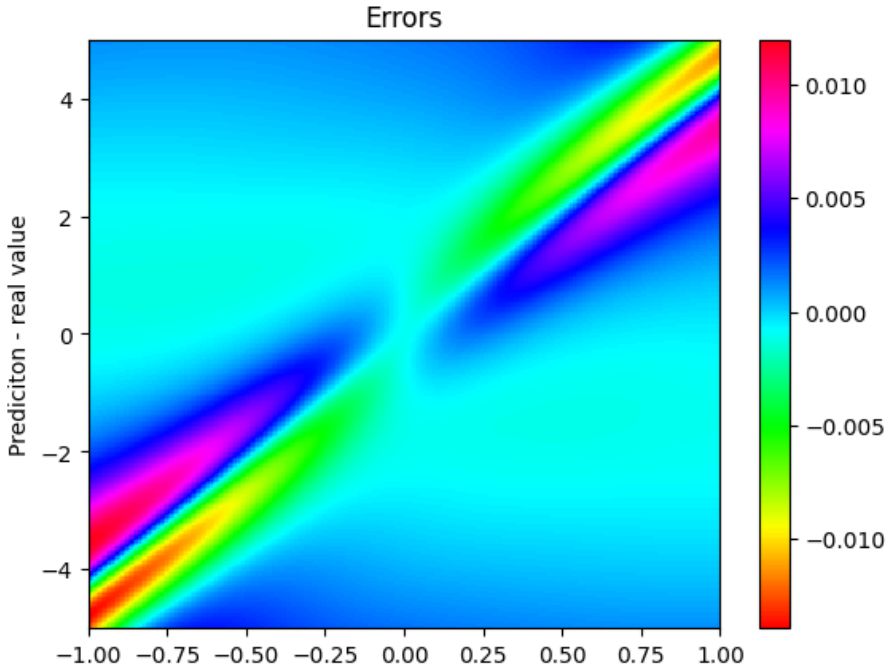


(b) Instantes posteriores à condição inicial.



Fonte: Elaborado pelo autor.

Figura 23 – Erro absoluto da solução do tipo 1 sóliton no caso com 4 camadas ocultas de 100 neurônios cada.



Fonte: Elaborado pelo autor.

4.2.1.2 Solução de dois sólitons da equação KdV via redes neurais

De forma semelhante e com a mesma arquitetura, a rede neural foi treinada para o caso de dois sólitons, onde, desta vez, a condição inicial referente ao caso de dois sólitons foi fornecida à máquina, de acordo com a equação (3.8)

$$u_t + 6uu_x + u_{xxx} = 0, \quad x \in [-5,5], \quad t \in [-1,1],$$

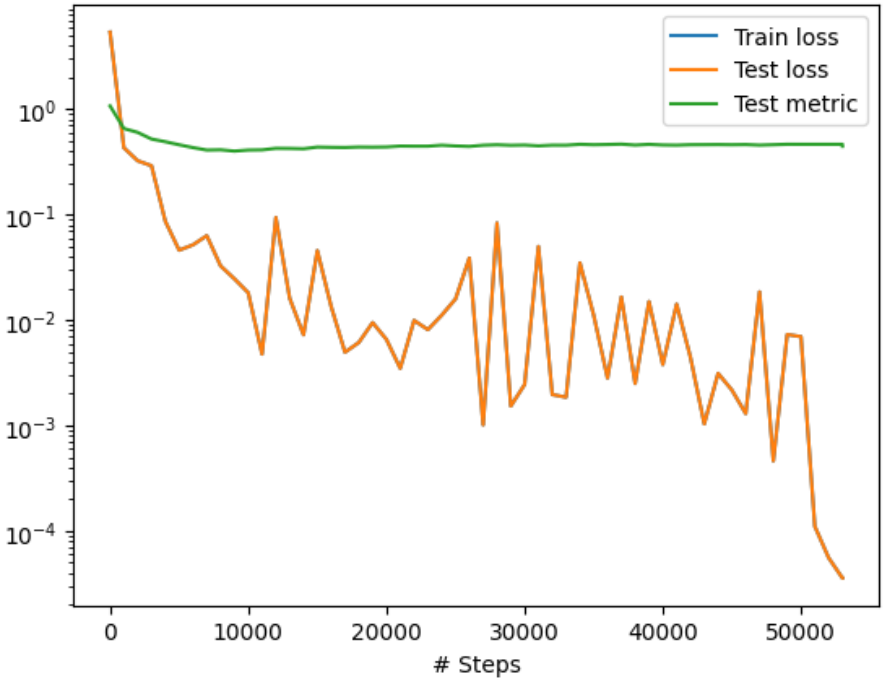
$$u(0,x) = 6\,sech^2(x).$$

A evolução da função de custo ao longo do treinamento se deu como na Figura 24, e os resultados encontrados se deram na Figura 25.

Pode ser observado que dois sólitons foram encontrados pela rede neural, porém, a partir da coordenada temporal, ao assumir valores com distância maior do que 0,2 da origem $t = 0$, onde as condições iniciais foram impostas, a rede neural obteve resultados distantes do esperado.

Uma vez calculado o erro relativo L2, foi encontrado o valor de $5{,}010 \times 10^{-1}$, que ao se tratar de um erro relativo, representa cerca de 50%!, ou seja, em média a solução não foi satisfatória. O erro da forma, diferença do valor real e valor predito, pôde ser plotado na Figura 26.

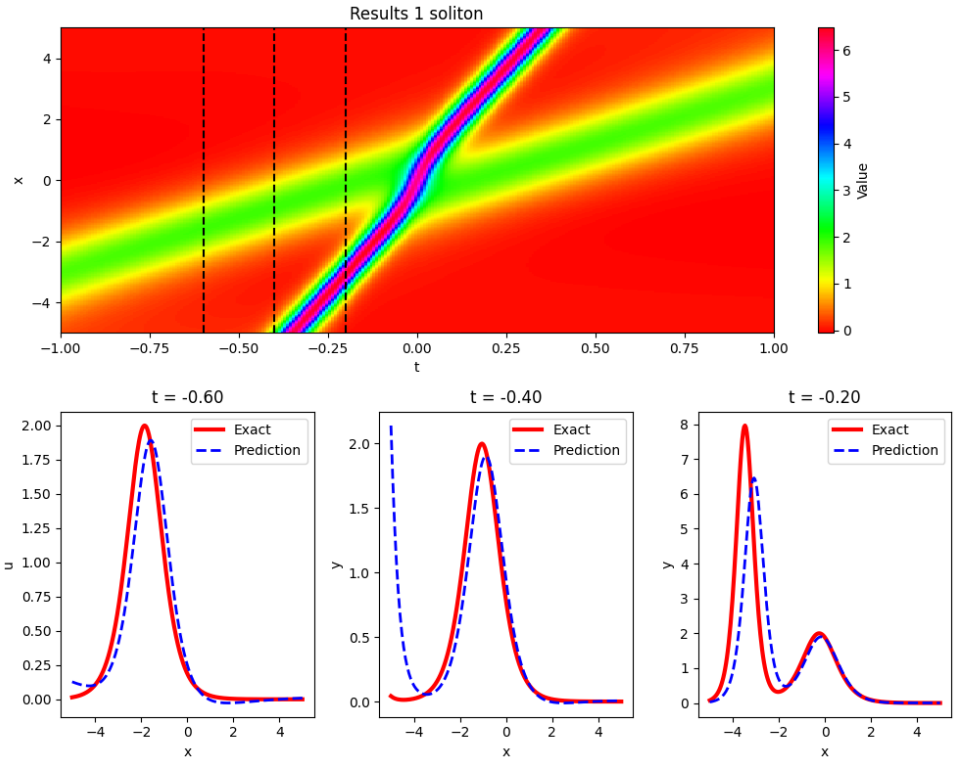
Figura 24 – Evolução do treinamento da rede neural, caso 2 sólitons



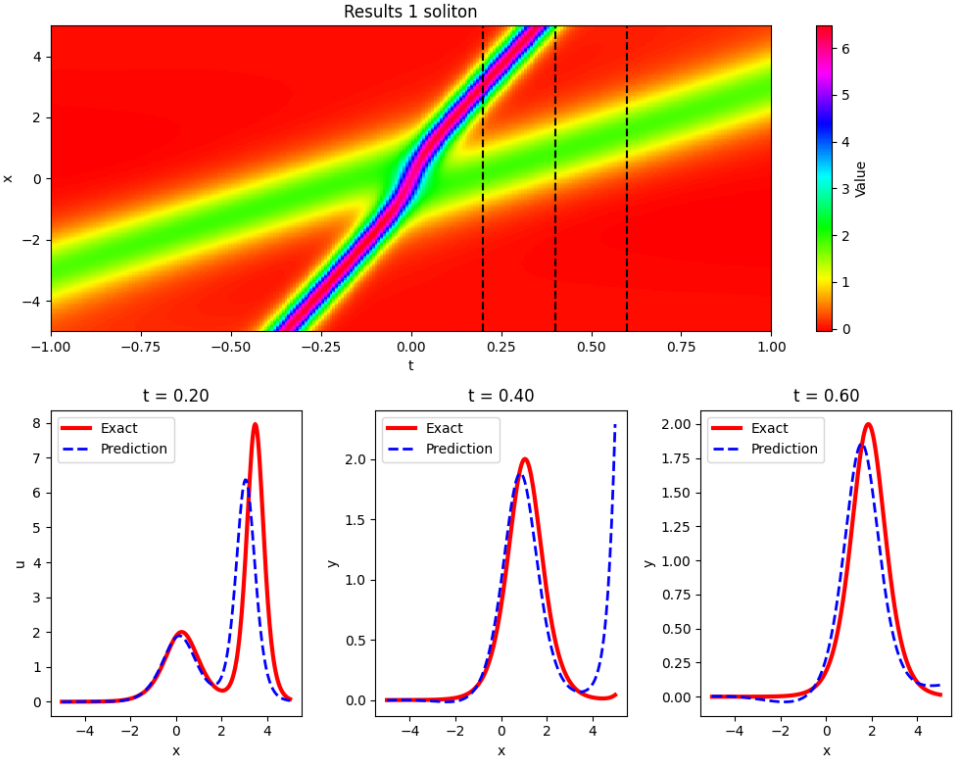
Fonte: Elaborado pelo autor.

Figura 25 – Resultado do treinamento de dois sólitons

(a) Instantes anteriores à condição inicial.

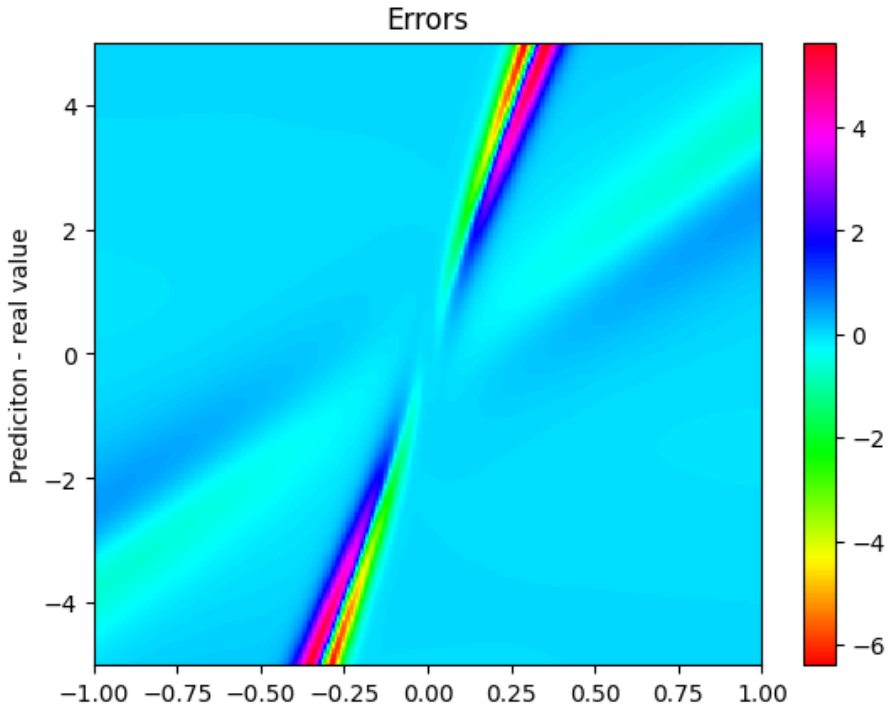


(b) Instantes posteriores à condição inicial.



Fonte: Elaborado pelo autor.

Figura 26 – Erro absoluto da solução do tipo 2 sólitons no caso com 4 camadas ocultas de 100 neurônios cada.



Fonte: Elaborado pelo autor.

Conforme observado na Figura 26, os erros não foram aceitáveis, ao apresentarem margens de erro da mesma ordem de grandeza que a solução. Representando um erro absoluto de em torno de 0,6 (que representaria 10% do valor da solução) apenas em torno da origem temporal da solução. Isso indica um treinamento e arquitetura não otimizados, sugerindo a necessidade de uma melhor escolha dos hiperparâmetros da rede.

Após identificar esses erros, foi realizado um processo sistemático de otimização da arquitetura da rede utilizada.

4.2.2 Testes sistemáticos da equação KdV

Como abordado na sessão de metodologia, o número de camadas ocultas testado variou de 1 a 10, sendo tais camadas com número de neurônios variando de 20 a 120, em passos de 20. Os resultados obtidos estão representados nas Tabelas 2, 3. Também foi testado o caso de uma condição inicial diferente, observando os impactos no treinamento da rede.

Ao se analisar as Tabelas 2 e 3, observou-se que o melhor resultado para o caso de um sóliton se deu para a rede com 8 camadas de 20 neurônios, que também pode ser notado na Figura 27).

No caso de dois sólitons, o melhor resultado se deu com o uso de 2 camadas de 40 neurônios, sendo os resultados dos testes melhor visualizados na Figura 28.

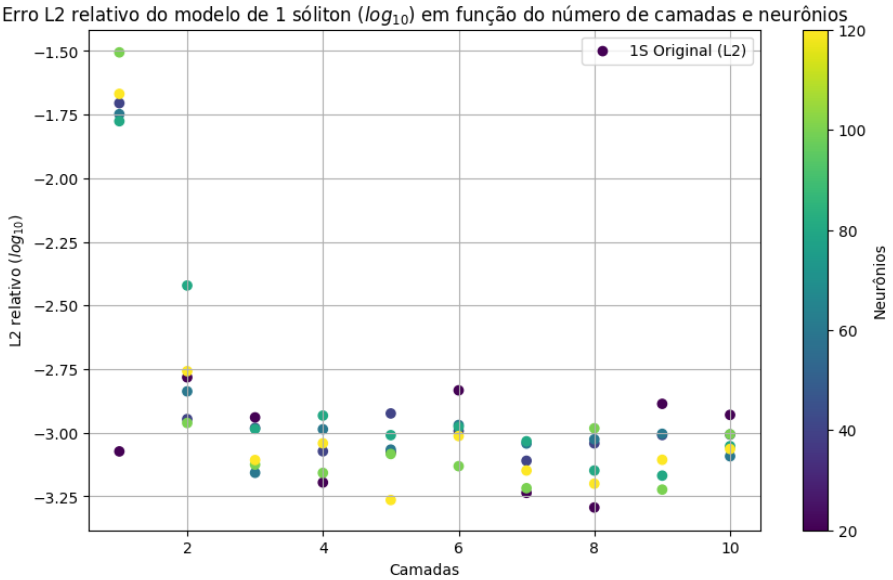
Tabela 2 – Resultados dos testes - Parte 1

Camadas	Neurônios	1S Original (L2)	2S Original (L2)	2S, IC em $t_0 = -0{,}2$
1	20	$8{,}450 \times 10^{-4}$	$9{,}035 \times 10^{-1}$	$8{,}324 \times 10^{-1}$
1	40	$1{,}972 \times 10^{-2}$	$9{,}066 \times 10^{-1}$	$8{,}821 \times 10^{-1}$
1	60	$1{,}785 \times 10^{-2}$	$9{,}081 \times 10^{-1}$	$8{,}796 \times 10^{-1}$
1	80	$1{,}675 \times 10^{-2}$	$8{,}944 \times 10^{-1}$	$8{,}445 \times 10^{-1}$
1	100	$3{,}122 \times 10^{-2}$	$9{,}144 \times 10^{-1}$	$8{,}544 \times 10^{-1}$
1	120	$2{,}145 \times 10^{-2}$	$9{,}001 \times 10^{-1}$	$8{,}515 \times 10^{-1}$
2	20	$1{,}651 \times 10^{-3}$	$1{,}011 \times 10^{-1}$	$1{,}338 \times 10^{-1}$
2	40	$1{,}132 \times 10^{-3}$	$5{,}544 \times 10^{-2}$	$2{,}131 \times 10^{-1}$
2	60	$1{,}454 \times 10^{-3}$	$7{,}204 \times 10^{-2}$	$1{,}283 \times 10^{-1}$
2	80	$3{,}789 \times 10^{-3}$	$6{,}756 \times 10^{-2}$	$1{,}636 \times 10^{-1}$
2	100	$1{,}093 \times 10^{-3}$	$6{,}321 \times 10^{-2}$	$1{,}951 \times 10^{-1}$
2	120	$1{,}746 \times 10^{-3}$	$6{,}681 \times 10^{-2}$	$4{,}037 \times 10^{-1}$
3	20	$1{,}149 \times 10^{-3}$	$1{,}910 \times 10^{-1}$	$2{,}121 \times 10^{-1}$
3	40	$1{,}045 \times 10^{-3}$	$8{,}955 \times 10^{-2}$	$2{,}749 \times 10^{-1}$
3	60	$6{,}964 \times 10^{-4}$	$8{,}225 \times 10^{-2}$	$2{,}293 \times 10^{-1}$
3	80	$1{,}035 \times 10^{-3}$	$8{,}022 \times 10^{-2}$	$4{,}459 \times 10^{-1}$
3	100	$7{,}500 \times 10^{-4}$	$1{,}674 \times 10^{-1}$	$2{,}628 \times 10^{-1}$
3	120	$7{,}815 \times 10^{-4}$	$2{,}348 \times 10^{-1}$	$5{,}008 \times 10^{-1}$
4	20	$6{,}379 \times 10^{-4}$	$1{,}485 \times 10^{-1}$	$3{,}330 \times 10^{-1}$
4	40	$8{,}459 \times 10^{-4}$	$2{,}321 \times 10^{-1}$	$4{,}758 \times 10^{-1}$
4	60	$1{,}032 \times 10^{-3}$	$2{,}039 \times 10^{-1}$	$4{,}884 \times 10^{-1}$
4	80	$1{,}168 \times 10^{-3}$	$3{,}909 \times 10^{-1}$	$4{,}871 \times 10^{-1}$
4	100	$6{,}956 \times 10^{-4}$	$3{,}821 \times 10^{-1}$	$5{,}465 \times 10^{-1}$
4	120	$9{,}092 \times 10^{-4}$	$2{,}964 \times 10^{-1}$	$5{,}602 \times 10^{-1}$
5	20	$8{,}385 \times 10^{-4}$	$1{,}237 \times 10^{-1}$	$4{,}128 \times 10^{-1}$
5	40	$1{,}191 \times 10^{-3}$	$3{,}700 \times 10^{-1}$	$2{,}637 \times 10^{-1}$
5	60	$8{,}588 \times 10^{-4}$	$2{,}441 \times 10^{-1}$	$4{,}078 \times 10^{-1}$
5	80	$9{,}789 \times 10^{-4}$	$3{,}500 \times 10^{-1}$	$5{,}685 \times 10^{-1}$
5	100	$8{,}254 \times 10^{-4}$	$2{,}724 \times 10^{-1}$	$5{,}650 \times 10^{-1}$
5	120	$5{,}445 \times 10^{-4}$	$3{,}381 \times 10^{-1}$	$5{,}708 \times 10^{-1}$
6	20	$1{,}468 \times 10^{-3}$	$3{,}508 \times 10^{-1}$	$2{,}592 \times 10^{-1}$
6	40	$1{,}017 \times 10^{-3}$	$3{,}135 \times 10^{-1}$	$3{,}943 \times 10^{-1}$
6	60	$1{,}073 \times 10^{-3}$	$3{,}872 \times 10^{-1}$	$5{,}395 \times 10^{-1}$
6	80	$1{,}054 \times 10^{-3}$	$4{,}103 \times 10^{-1}$	$5{,}287 \times 10^{-1}$
6	100	$7{,}392 \times 10^{-4}$	$4{,}154 \times 10^{-1}$	$4{,}542 \times 10^{-1}$
6	120	$9{,}711 \times 10^{-4}$	$4{,}456 \times 10^{-1}$	$6{,}019 \times 10^{-1}$
7	20	$5{,}807 \times 10^{-4}$	$2{,}233 \times 10^{-1}$	$2{,}761 \times 10^{-1}$
7	40	$7{,}764 \times 10^{-4}$	$2{,}356 \times 10^{-1}$	$5{,}063 \times 10^{-1}$
7	60	$9{,}075 \times 10^{-4}$	$3{,}263 \times 10^{-1}$	$5{,}900 \times 10^{-1}$
7	80	$9{,}260 \times 10^{-4}$	$4{,}333 \times 10^{-1}$	$4{,}677 \times 10^{-1}$
7	100	$6{,}059 \times 10^{-4}$	$4{,}451 \times 10^{-1}$	$5{,}800 \times 10^{-1}$
7	120	$7{,}109 \times 10^{-4}$	$4{,}676 \times 10^{-1}$	$5{,}971 \times 10^{-1}$

Tabela 3 – Resultados dos testes - Parte 2

Camadas	Neurônios	1S Original (L2)	2S Original (L2)	2S, IC em $t_0 = -0{,}2$
8	20	$5{,}087 \times 10^{-4}$	$2{,}998 \times 10^{-1}$	$2{,}411 \times 10^{-1}$
8	40	$9{,}089 \times 10^{-4}$	$2{,}674 \times 10^{-1}$	$5{,}305 \times 10^{-1}$
8	60	$9{,}427 \times 10^{-4}$	$4{,}420 \times 10^{-1}$	$5{,}348 \times 10^{-1}$
8	80	$7{,}107 \times 10^{-4}$	$4{,}877 \times 10^{-1}$	$5{,}047 \times 10^{-1}$
8	100	$1{,}041 \times 10^{-3}$	$5{,}218 \times 10^{-1}$	$4{,}663 \times 10^{-1}$
8	120	$6{,}310 \times 10^{-4}$	$4{,}966 \times 10^{-1}$	$5{,}940 \times 10^{-1}$
9	20	$1{,}299 \times 10^{-3}$	$3{,}972 \times 10^{-1}$	$4{,}950 \times 10^{-1}$
9	40	$9{,}798 \times 10^{-4}$	$3{,}598 \times 10^{-1}$	$4{,}833 \times 10^{-1}$
9	60	$9{,}906 \times 10^{-4}$	$4{,}199 \times 10^{-1}$	$5{,}887 \times 10^{-1}$
9	80	$6{,}787 \times 10^{-4}$	$4{,}864 \times 10^{-1}$	$5{,}933 \times 10^{-1}$
9	100	$5{,}981 \times 10^{-4}$	$5{,}204 \times 10^{-1}$	$4{,}506 \times 10^{-1}$
9	120	$7{,}832 \times 10^{-4}$	$5{,}262 \times 10^{-1}$	$5{,}914 \times 10^{-1}$
10	20	$1{,}176 \times 10^{-3}$	$4{,}102 \times 10^{-1}$	$2{,}739 \times 10^{-1}$
10	40	$9{,}844 \times 10^{-4}$	$4{,}027 \times 10^{-1}$	$4{,}328 \times 10^{-1}$
10	60	$8{,}088 \times 10^{-4}$	$4{,}479 \times 10^{-1}$	$5{,}713 \times 10^{-1}$
10	80	$8{,}863 \times 10^{-4}$	$5{,}353 \times 10^{-1}$	$4{,}094 \times 10^{-1}$
10	100	$9{,}869 \times 10^{-4}$	$5{,}187 \times 10^{-1}$	$9{,}129 \times 10^{-1}$
10	120	$8{,}637 \times 10^{-4}$	$4{,}799 \times 10^{-1}$	$6{,}224 \times 10^{-1}$

Figura 27 – Gráfico do erro L2 relativo (log_{10}) em relação ao número de camadas e neurônios da rede neural, caso de 1 sóliton.



Fonte: Elaborado pelo autor.

Figura 28 – Gráfico do erro L2 relativo (log_{10}) em relação ao número de camadas e neurônios da rede neural, caso de 2 sólitons.

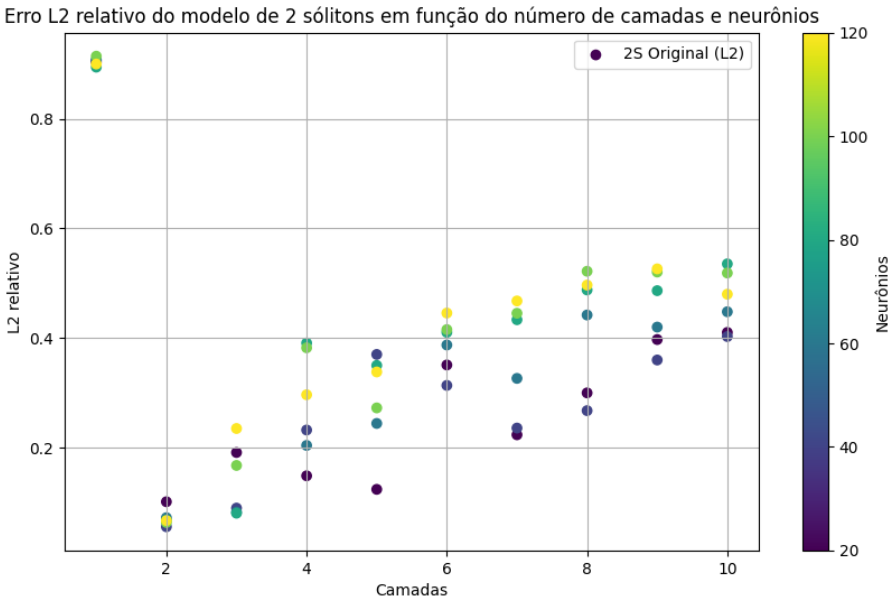
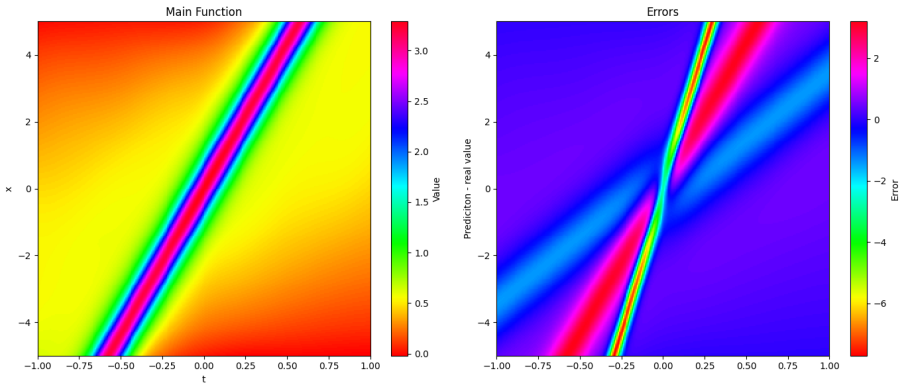


Figura 29 – Modelo com condição inicial relativa a dois sólitons, com uma camada oculta de 120 neurônios.



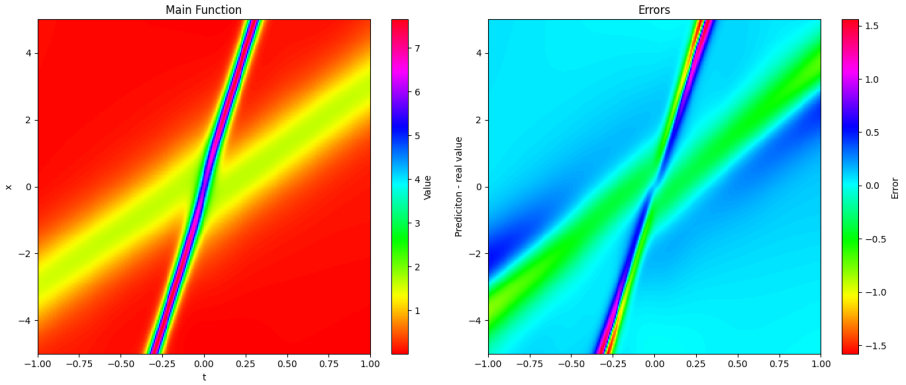
Fonte: Elaborado pelo autor.

Uma análise mais criteriosa foi realizada no caso de dois sólitons, considerando a arquitetura mínima necessária para distingui-los e, separadamente, o uso de vínculos adicionais.

4.2.2.1 Arquitetura mínima para a distinção de dois sólitons pela máquina

Um fenômeno interessante observado no caso de dois sólitons foi uma mudança brusca nos erros L2 relativos ao se mudar de uma camada para duas camadas: $9{,}001 \times 10^{-1}$ para $1{,}011 \times 10^{-1}$, ao se mudar de 1 camada com 120 neurônios, para 2 camadas de 20 neurônios. Observando os resultados dos dois modelos, para 1 camada de 120 neurônios, (representado na Figura 29), o resultado encontrado pela rede parece contemplar um sóliton intermediário entre os dois sólitons pertinentes à equação diferencial.

Figura 30 – Modelo com condição inicial relativa a dois sólitons, com duas camadas ocultas de 20 neurônios.



Fonte: Elaborado pelo autor.

Em seguida, para o caso referente a um modelo com 2 camadas ocultas de 20 neurônios cada, representado na Figura 30, tem-se o início da representação de dois sólitons, mesmo que com erros ainda altos.

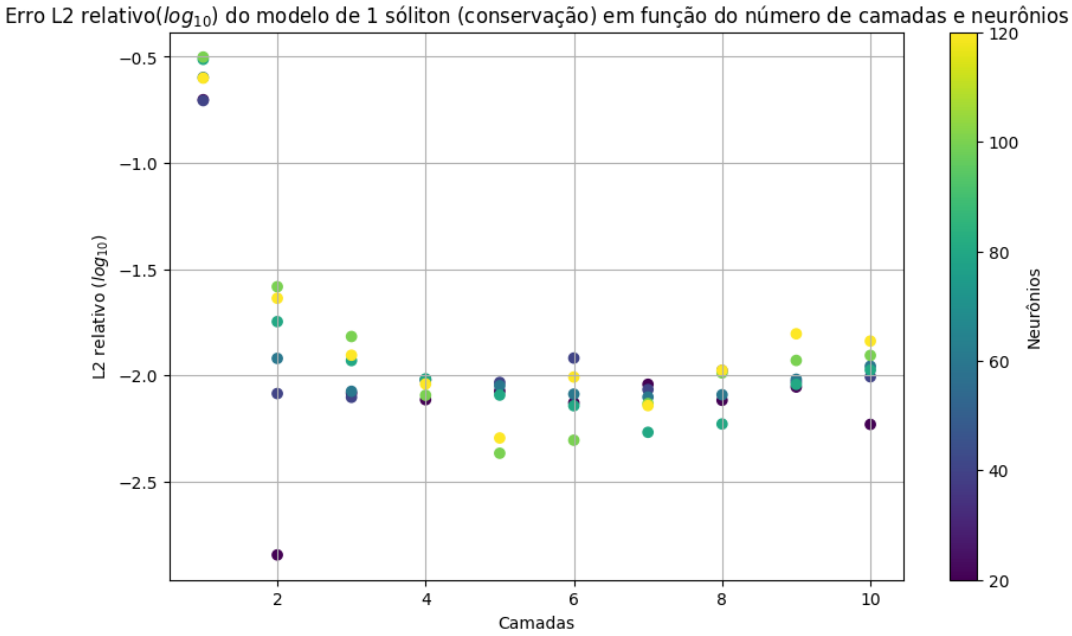
Tal resultado sugere a possibilidade de estudo de uma arquitetura mínima para a representação do problema físico em questão. Como este foi representado pela máquina, é sugerida a necessidade mínima de duas camadas para a distinção de dois sólitons para a rede neural, sendo que antes deste mínimo o resultado se assimila a um sóliton intermediário aos dois sólitons buscados.

A condição inicial foi fornecida no instante em que os dois sólitons encontram-se exatamente na mesma posição, conforme descrito pela solução exata da equação de KdV para $t = 0$. Embora o princípio clássico de superposição não seja aplicável devido à natureza não linear da equação, essa configuração inicial representa a interação de dois sólitons, a princípio num momento em que estes não são distinguíveis [1].

O impacto do posicionamento da condição inicial em um ponto onde a separação dos sólitons é mais clara ($t_0 = 0{,}2$) também foi analisado. Os resultados, apresentados nas Tabelas 2 e 3, indicaram que, nos casos em que a rede não distinguiu a existência de dois sólitons, houve uma leve melhoria. No entanto, para os casos de arquiteturas suficientemente robustas, observou-se uma deterioração significativa nos resultados, sugerindo que uma melhor generalização dos resultados poderia ser obtida com uma condição inicial em $t = 0$, mesmo sem uma distinção clara quanto à existência de dois sólitons. Isso sugere que os vínculos matemáticos impostos pela condição inicial e pela equação diferencial são suficientes para permitir a distinção de dois sólitons da equação KdV pela rede neural, desde que a arquitetura seja suficientemente robusta.

[1] Como a amplitude dos sólitons está diretamente relacionada às suas velocidades de propagação, as soluções do tipo sóliton da equação KdV devem obedecer a relações matemáticas bem definidas. Dessa forma, uma condição inicial composta por dois sólitons, mesmo em um instante de coincidência espacial, carrega essas informações características.

Figura 31 – Gráfico do erro L2 relativo (log_{10}) em relação ao número de camadas e neurônios da rede neural, caso de 1 sóliton em que o vínculo extra relativo a segunda lei de conservação foi utilizado.



Fonte: Elaborado pelo autor.

Em seguida serão apresentados os resultados ao se utilizar vínculos extras referentes às leis de conservação da equação KdV.

4.2.2.2 Caso de vinculo extra relativo a leis de conservação

Foi testado sistematicamente também o uso de um vínculo extra, sendo este a segunda lei de conservação da equação KdV, tratado na Seção 3.2. Os resultados obtidos foram representados nas Tabelas 4 e 5.

Foi observado uma diferença nas arquiteturas que atingiram os melhores resultados. No caso de 1 sóliton: 2 camadas de 20 neurônios (Figura 31), e no caso de dois sólitons: 5 camadas de 120 neurônios (Figura 31).

A implementação do vínculo extra relacionado à segunda lei de conservação obteve resultados inferiores ao caso sem vínculo, em que apenas a equação diferencial foi utilizada. Logo, essa implementação não foi utilizada no modelo final para a solução do tipo 1 e 2 sólitons da equação KdV. Embora a implementação tenha se mostrado ineficaz, o método de implementação de vínculos extras foi estudado, demonstrando a viabilidade desse tipo de abordagem e a forma de aplicá-la no estudo de equações diferenciais não lineares.

Tabela 4 – Resultados dos testes com o uso de leis de conservação - Parte 1

Camadas	Neurônios	1S Conservação (L2)	2S Conservação (L2)
1	20	$1{,}984 \times 10^{-1}$	$9{,}314 \times 10^{-1}$
1	40	$1{,}965 \times 10^{-1}$	$9{,}267 \times 10^{-1}$
1	60	$2{,}525 \times 10^{-1}$	$9{,}277 \times 10^{-1}$
1	80	$3{,}073 \times 10^{-1}$	$9{,}160 \times 10^{-1}$
1	100	$3{,}150 \times 10^{-1}$	$9{,}110 \times 10^{-1}$
1	120	$2{,}508 \times 10^{-1}$	$9{,}120 \times 10^{-1}$
2	20	$1{,}429 \times 10^{-3}$	$8{,}006 \times 10^{-1}$
2	40	$8{,}213 \times 10^{-3}$	$8{,}133 \times 10^{-1}$
2	60	$1{,}202 \times 10^{-2}$	$8{,}167 \times 10^{-1}$
2	80	$1{,}791 \times 10^{-2}$	$8{,}324 \times 10^{-1}$
2	100	$2{,}615 \times 10^{-2}$	$8{,}332 \times 10^{-1}$
2	120	$2{,}307 \times 10^{-2}$	$8{,}584 \times 10^{-1}$
3	20	$8{,}235 \times 10^{-3}$	$7{,}575 \times 10^{-1}$
3	40	$7{,}887 \times 10^{-3}$	$7{,}900 \times 10^{-1}$
3	60	$8{,}415 \times 10^{-3}$	$7{,}895 \times 10^{-1}$
3	80	$1{,}174 \times 10^{-2}$	$8{,}035 \times 10^{-1}$
3	100	$1{,}524 \times 10^{-2}$	$7{,}885 \times 10^{-1}$
3	120	$1{,}244 \times 10^{-2}$	$8{,}016 \times 10^{-1}$
4	20	$7{,}684 \times 10^{-3}$	$7{,}760 \times 10^{-1}$
4	40	$9{,}509 \times 10^{-3}$	$7{,}867 \times 10^{-1}$
4	60	$9{,}543 \times 10^{-3}$	$7{,}822 \times 10^{-1}$
4	80	$9{,}629 \times 10^{-3}$	$7{,}871 \times 10^{-1}$
4	100	$8{,}063 \times 10^{-3}$	$7{,}755 \times 10^{-1}$
4	120	$9{,}113 \times 10^{-3}$	$7{,}522 \times 10^{-1}$
5	20	$8{,}435 \times 10^{-3}$	$8{,}093 \times 10^{-1}$
5	40	$9{,}267 \times 10^{-3}$	$8{,}025 \times 10^{-1}$
5	60	$8{,}940 \times 10^{-3}$	$7{,}740 \times 10^{-1}$
5	80	$8{,}078 \times 10^{-3}$	$7{,}722 \times 10^{-1}$
5	100	$4{,}301 \times 10^{-3}$	$7{,}217 \times 10^{-1}$
5	120	$5{,}075 \times 10^{-3}$	$6{,}632 \times 10^{-1}$
6	20	$7{,}435 \times 10^{-3}$	$8{,}035 \times 10^{-1}$
6	40	$1{,}206 \times 10^{-2}$	$8{,}034 \times 10^{-1}$
6	60	$8{,}163 \times 10^{-3}$	$7{,}794 \times 10^{-1}$
6	80	$7{,}197 \times 10^{-3}$	$7{,}200 \times 10^{-1}$
6	100	$4{,}953 \times 10^{-3}$	$7{,}027 \times 10^{-1}$
6	120	$9{,}823 \times 10^{-3}$	$6{,}642 \times 10^{-1}$
7	20	$9{,}082 \times 10^{-3}$	$8{,}026 \times 10^{-1}$
7	40	$8{,}578 \times 10^{-3}$	$7{,}962 \times 10^{-1}$
7	60	$7{,}911 \times 10^{-3}$	$7{,}631 \times 10^{-1}$
7	80	$5{,}397 \times 10^{-3}$	$7{,}163 \times 10^{-1}$
7	100	$7{,}365 \times 10^{-3}$	$6{,}965 \times 10^{-1}$
7	120	$7{,}208 \times 10^{-3}$	$6{,}742 \times 10^{-1}$

Tabela 5 – Resultados dos testes com o uso de leis de conservação - Parte 2

Camadas	Neurônios	1S Conservação (L2)	2S Conservação (L2)
8	20	$7{,}634 \times 10^{-3}$	$8{,}182 \times 10^{-1}$
8	40	$1{,}055 \times 10^{-2}$	$8{,}129 \times 10^{-1}$
8	60	$8{,}107 \times 10^{-3}$	$7{,}488 \times 10^{-1}$
8	80	$5{,}906 \times 10^{-3}$	$7{,}460 \times 10^{-1}$
8	100	$1{,}027 \times 10^{-2}$	$7{,}145 \times 10^{-1}$
8	120	$1{,}057 \times 10^{-2}$	$7{,}167 \times 10^{-1}$
9	20	$8{,}818 \times 10^{-3}$	$8{,}340 \times 10^{-1}$
9	40	$9{,}327 \times 10^{-3}$	$7{,}765 \times 10^{-1}$
9	60	$9{,}585 \times 10^{-3}$	$7{,}303 \times 10^{-1}$
9	80	$9{,}074 \times 10^{-3}$	$7{,}432 \times 10^{-1}$
9	100	$1{,}176 \times 10^{-2}$	$7{,}479 \times 10^{-1}$
9	120	$1{,}570 \times 10^{-2}$	$7{,}199 \times 10^{-1}$
10	20	$5{,}879 \times 10^{-3}$	$8{,}552 \times 10^{-1}$
10	40	$9{,}855 \times 10^{-3}$	$8{,}012 \times 10^{-1}$
10	60	$1{,}105 \times 10^{-2}$	$7{,}777 \times 10^{-1}$
10	80	$1{,}059 \times 10^{-2}$	$7{,}853 \times 10^{-1}$
10	100	$1{,}242 \times 10^{-2}$	$7{,}785 \times 10^{-1}$
10	120	$1{,}451 \times 10^{-2}$	$7{,}016 \times 10^{-1}$

4.2.3 Treinamento robusto a partir da arquitetura de melhor resultado

Uma vez que foram determinadas as arquiteturas com melhor aproveitamento para o problema, os testes da implementação direta foram refeitos, em que a arquitetura adotada foi aquela de melhor desempenho obtida nos testes sistemáticos para cada caso.

4.2.3.1 Equação KdV com um sóliton

A equação diferencial e condição inicial pertinente a um sóliton, conforme a equação (3.6), foi implementada, sem vínculos extras, seguindo a arquitetura de rede

```
layer_size = [2] + [20] * 8 + [1]
```

com 8 camadas ocultas de 20 neurônios cada. Para o caso de 1 sóliton, após 50 mil épocas de treinamento com otimizador Adam, a taxa de aprendizado foi reduzida gradativamente utilizando um algoritmo de decaimento inverso com o tempo, o modelo foi treinado por mais 100 mil épocas, e por fim, o resultado foi otimizado utilizando o algoritmo L-BFGS. A evolução da função de custo ao longo do processo de treinamento está representada na Figura 33.

Os resultados da resposta do modelo são apresentados de forma gráfica na Figura 34.

Figura 32 – Gráfico do erro L2 relativo (log_{10}) em relação ao número de camadas e neurônios da rede neural, caso de 2 sólitons em que o vínculo extra relativo a segunda lei de conservação foi utilizado.

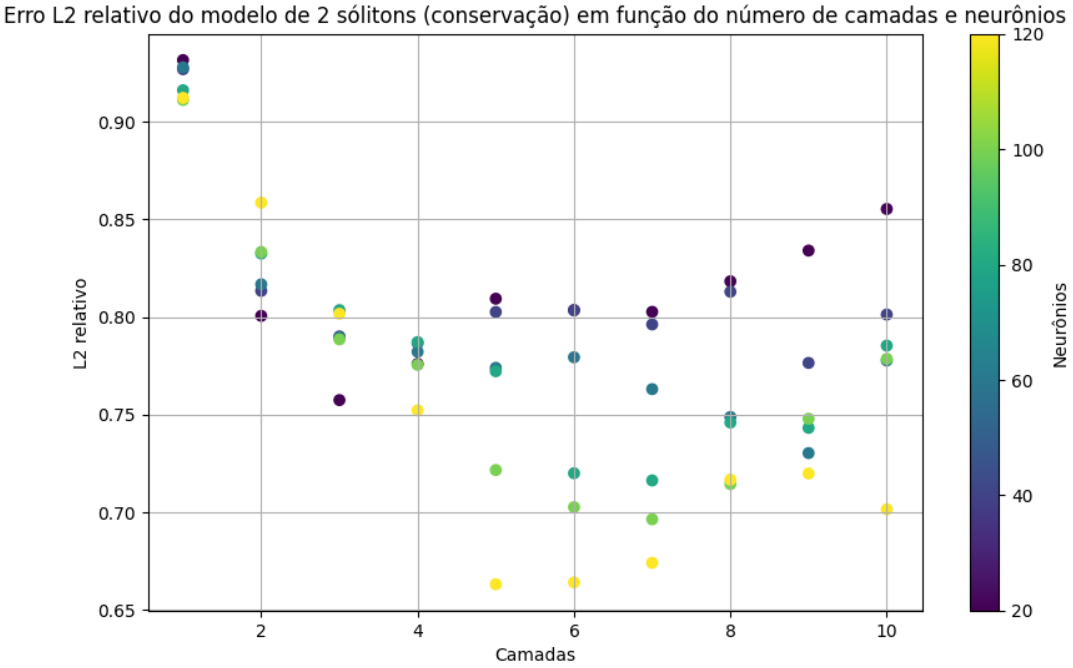
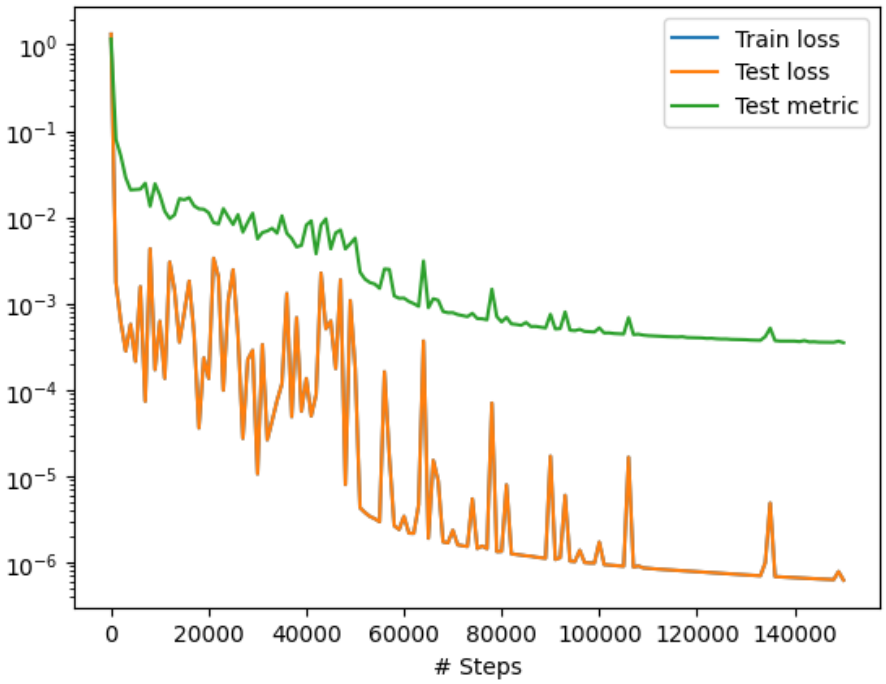


Figura 33 – Evolução do treinamento da rede neural, caso 1 sóliton em que foi realizado um treinamento mais robusto.



Fonte: Elaborado pelo autor.

Os erros da saída do modelo com relação a resposta exata são apresentados na Figura 35.

O erro L2 relativo médio final encontrado foi de $3{,}760 \times 10^{-4}$, menor que o obtido no teste sistemático e uma ordem de grandeza abaixo do caso de implementação direta.

4.2.3.2 Equação KdV com dois sólitons

De forma semelhante e com a arquitetura de melhor desempenho obtida pelos testes sistemáticos, realizou-se o treinamento da rede para o caso da equação KdV com dois sólitons. A arquitetura da rede é apresentada abaixo.

```
layer_size = [2] + [40] * 2 + [1]
```

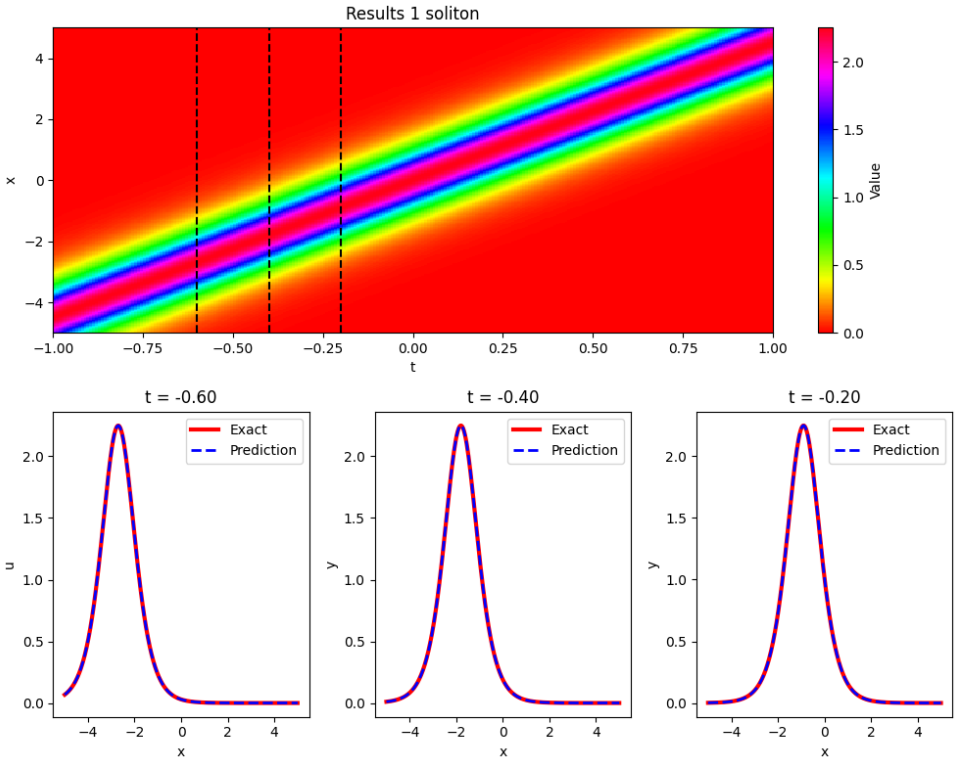
Com duas camadas ocultas de 40 neurônios cada, a rede neural foi treinada para o caso de dois sólitons, de acordo com a equação (3.8).

O processo de treinamento contou com 500 mil épocas utilizando o otimizador Adam, seguido de uma otimização com o algoritmo L-BFGS. A evolução do treinamento se deu como na figura 36, e os resultados encontrados são apresentados na na Figura 37.

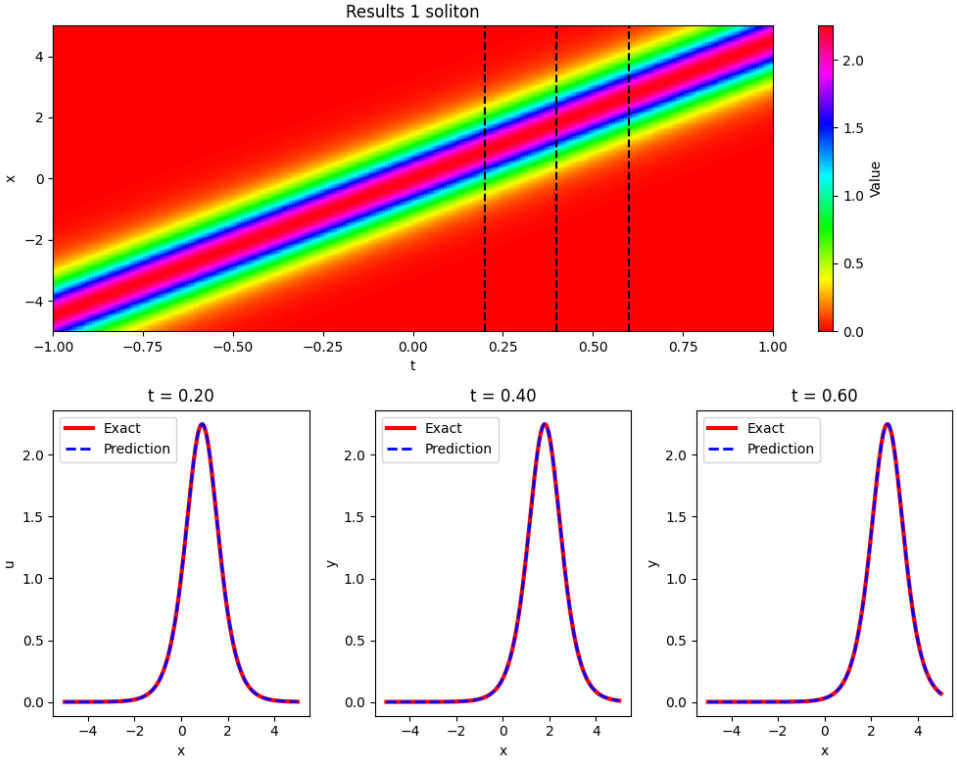
Uma vez calculado o erro relativo L_2, foi encontrado o valor de $1{,}530 \times 10^{-2}$, que é uma ordem de grandeza inferior à implementação direta e 5 vezes menor do que o melhor caso dos testes sistemáticos. Para uma melhor visualização, os erros entre o valor real e o valor predito são apresentados na Figura 38, onde, dessa vez, observa-se um erro ainda presente, porém representando, em média, erros absolutos de $1{,}5 \times 10^{-2}$, dentro do domínio estudado, com erros significativamente maiores, da ordem de 10^{-1}, nas bordas do domínio.

Figura 34 – Resultado do treinamento mais robusto de um sóliton

(a) Instantes anteriores à condição inicial.

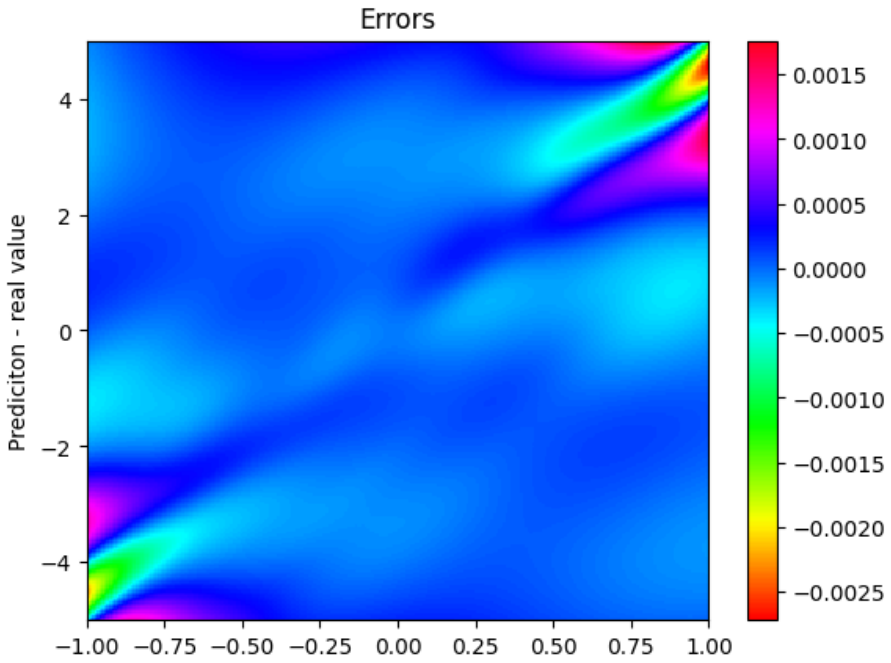


(b) Instantes posteriores à condição inicial.



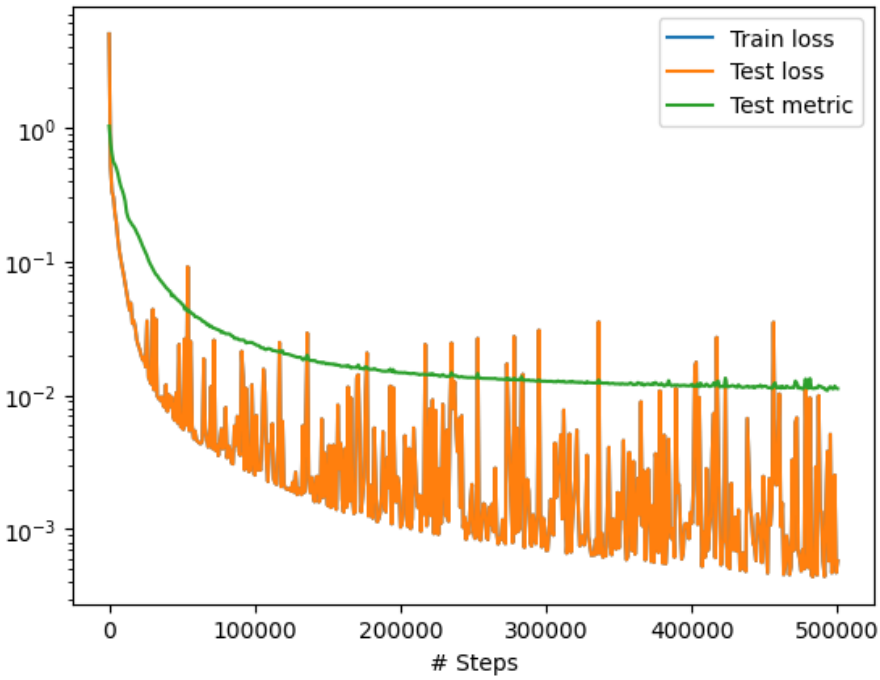
Fonte: Elaborado pelo autor.

Figura 35 – Erro absoluto da solução do tipo 1 sóliton no caso com 8 camadas ocultas de 20 neurônios cada.



Fonte: Elaborado pelo autor.

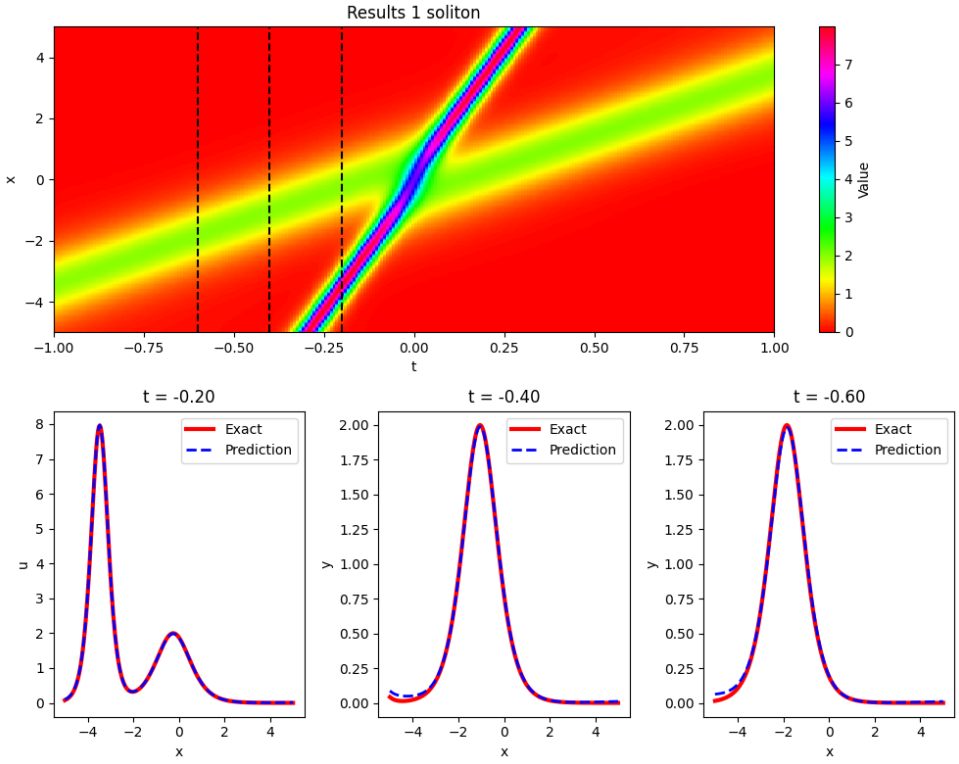
Figura 36 – Evolução do treinamento da rede neural, caso 2 sólitons



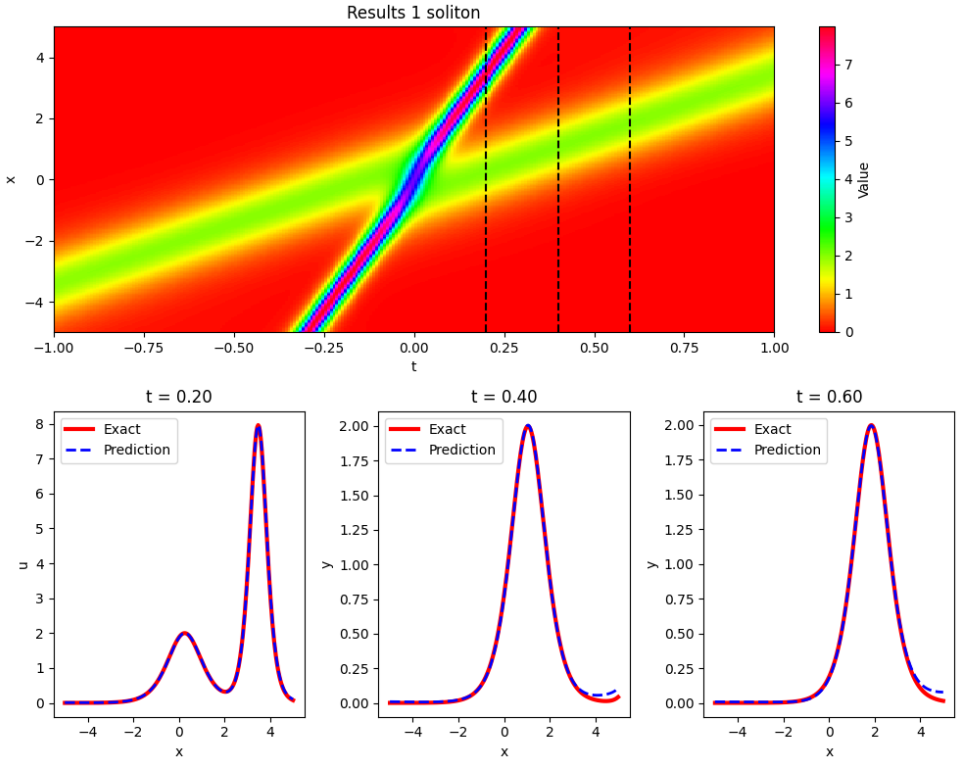
Fonte: Elaborado pelo autor.

Figura 37 – Resultado do treinamento de dois sólitons pelo modelo mais robusto e otimizado.

(a) Instantes anteriores à condição inicial.

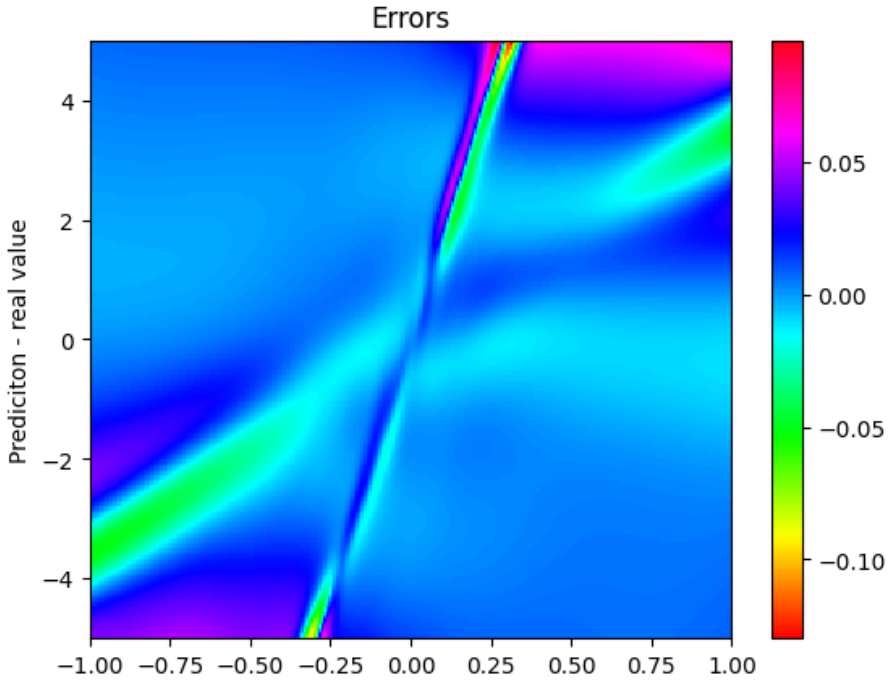


(b) Instantes posteriores à condição inicial.



Fonte: Elaborado pelo autor.

Figura 38 – Erro absoluto da solução do tipo 2 sólitons no caso com 2 camadas ocultas de 40 neurônios cada.



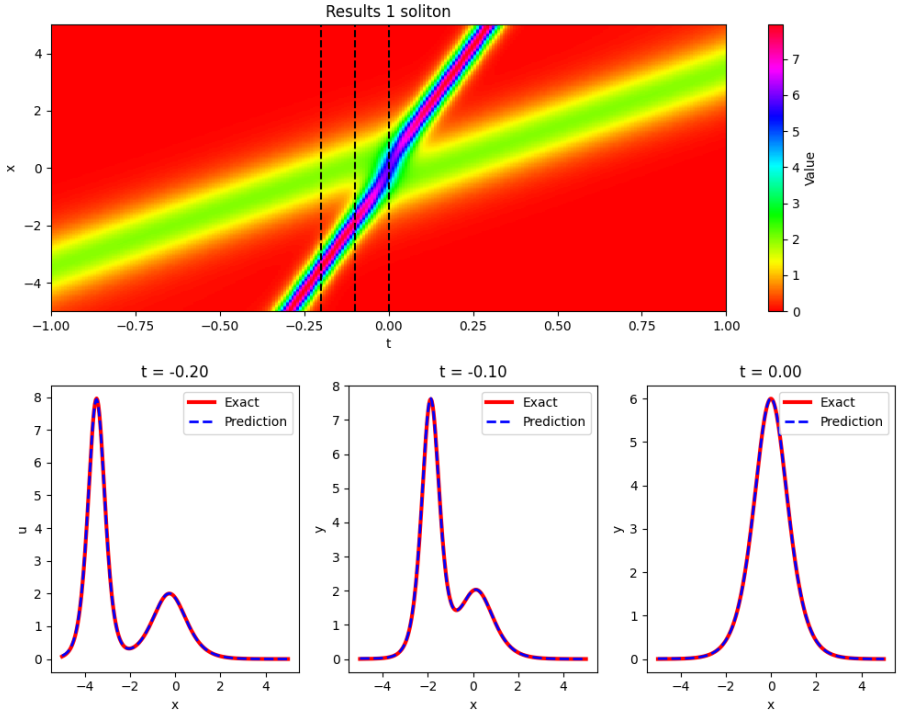
Fonte: Elaborado pelo autor.

Mesmo com um erro ainda observável, o fenômeno de colisão de dois sólitons da equação KdV pode ser observado na Figura 39. Na colisão de dois sólitons, observa-se que, antes da colisão, o sóliton de maior amplitude, devido à sua maior velocidade, aproxima-se e colide com o sóliton de menor amplitude. É observado que o princípio de superposição não se aplica ao sólitons, uma vez que não se tratam de ondas lineares, em que o resultado da colizão em t=0 é uma onda com amplitude intermediária entre os dois sólitons. Após a colisão, ambos os sólitons continuam seu movimento sem alteração em sua forma ou tamanho, a menos de uma diferença de fase.

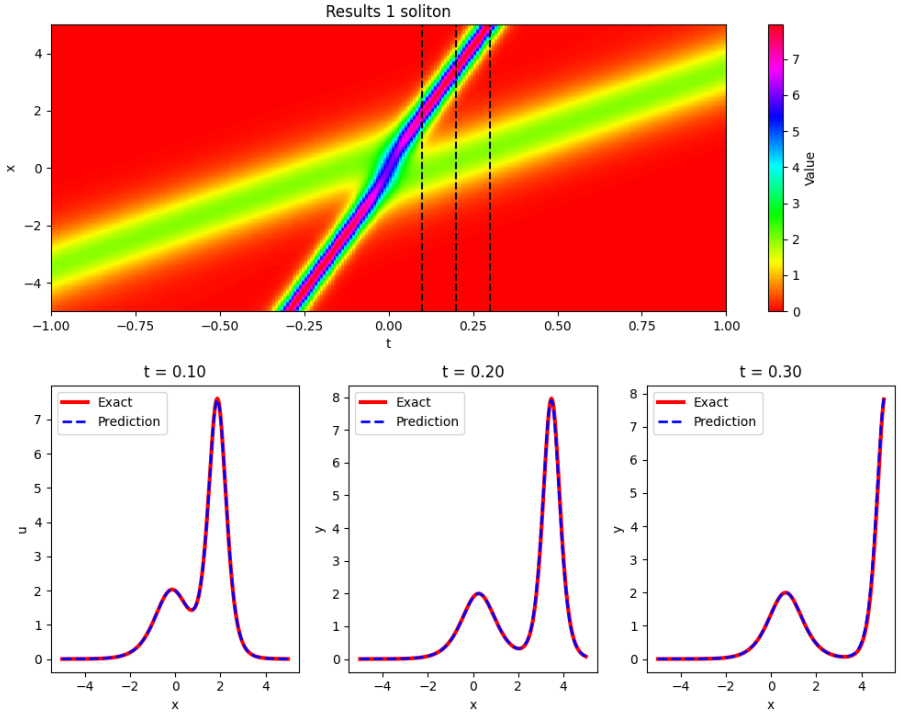
Uma possível forma de melhorar os erros seria a implementação de condições de contorno extras, afinal, foram utilizadas somente condições iniciais em que foram fornecidas 100 amostras da solução, que foram impostas em $t = 0$. Mesmo com somente esta quantidade de dados, e somente uma condição inicial, foi demonstrada uma solução de uma equação diferencial não linear com o método das PINNs em um caso mais complexo envolvendo colisões de dois sólitons, com um erro razoável.

Figura 39 – Colisão de dois sólitons: Na colisão de dois sólitons, observa-se que, antes da colisão, o sóliton de maior amplitude, devido à sua maior velocidade, aproxima-se e colide com o sóliton de menor amplitude. Após a colisão, ambos os sólitons continuam seu movimento sem alteração em sua forma ou tamanho.

(a) Instantes anteriores e equivalentes à condição inicial.



(b) Instantes posteriores à condição inicial.



Fonte: Elaborado pelo autor.

4.3 Sine-Gordon

Na primeira parte deste capítulo, foi demonstrado o uso de redes neurais para a resolução de uma equação diferencial ordinária com coeficientes variáveis. Já na segunda parte, foi abordada a resolução de uma equação diferencial parcial não linear. Nesta Seção, será investigada a equação de Sine-Gordon, explorando a variação de um de seus parâmetros, m. O objetivo é não apenas obter uma solução específica, mas aprender uma família de soluções dentro de um intervalo especificado para o parâmetro m.

4.3.1 Sine-Gordon utilizando 3 condições de contorno

Num problema com três condições de contorno, mais especificamente, uma condição inicial e duas condições de contorno, tem-se a equação diferencial, domínio das coordenadas e do parâmetro m definida por

$$u_{tt} - u_{xx} + m^2 \sin u = 0, \quad m \in [m_0, m_f],\ x \in [-5, 5],\ t \in [0, 2]. \tag{4.1}$$

Para a equação (4.1) especificou-se a condição inicial e as condições de contorno

$$\begin{aligned} u(x, m, 0) &= 4 \arctan\left(e^{m\,\gamma\,x+\delta}\right), \\ u(x=-5, m, t) &= 4 \arctan\left(e^{m\gamma(-5-vt)+\delta}\right), \\ u(x=5, m, t) &= 4 \arctan\left(e^{m\gamma(5-vt)+\delta}\right). \end{aligned} \tag{4.2}$$

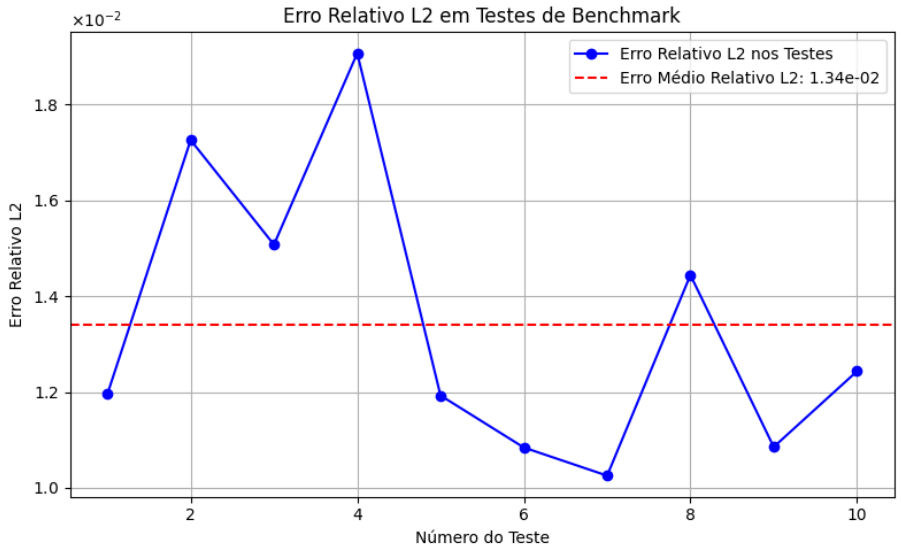
Em relação à equação (4.1), foi informado para fins de treinamento, 2000 pontos de colocação distribuídos aleatoriamente no domínio tridimensional $\{(x,m,t) \mid x \in [-5,5], m \in [m_0, m_f], t \in [0,2]\}$. Para as equações relacionadas às condições de contorno (4.2), foram informados à máquina 128 pontos da condição inicial, em que t foi fixado em $t = 0$, e x e m foram distribuídos aleatoriamente entre o domínio da variável x e intervalo de parâmetro m, assim como 128 pontos fixados para cada caso da variável $x = -5$ e $x = 5$, em que o parâmetro m e a variável t tiveram valores estipulados aleatoriamente, totalizando 384 pontos para as condições de contorno.

Para os demais parâmetros, presentes na equação diferencial e condições de contorno, foi utilizado $v = 0{,}65$ e $\delta = 0$, ainda γ foi definido conforme a equação (2.51). Foi feito inicialmente um teste de *benchmark*, em que foi fixado o valor de $m = 0{,}3$. Em seguida foi treinada a rede para variações a princípio pequenas de m, sendo estipulado um limite inferior de m com $m = 0{,}05$, uma vez que a equação diferencial representaria outro modelo no caso $m = 0$[2].

[2] Para $m = 0$, a equação de Sine-Gordon escrita de forma mais geral, é reduzida a equação de onda $u_{tt} = u_{xx}$, em unidades c=1.

Para o teste de com $m = 0{,}3$ foi utilizada uma rede com dois neurônios na primeira camada, representando as coordenadas x e t, de forma semelhante ao caso anterior da equação KdV. Uma vez que foi utilizado o otimizador Adam, que faz uso de técnicas estocásticas, é esperado um resultado diferente a cada treinamento. Com o intuito de estipular um valor de erro que representa modelo de *benchmark*, o calculo do erro foi feito durante 10 treinamentos distintos, em que o resultado na forma de erro L2 relativo, foi representado na Figura 40.

Figura 40 – Resultados do modelo da equação de Sine-Gordon com o parâmetro m fixado em 0,3 treinado por 20 mil épocas, com refinamento posterior via otimização L-BFGS.



O valor médio obtido nos 10 treinamentos para o cálculo do erro L2 relativo no modelo de *benchmark* foi de $1{,}341 \times 10^{-2}$.

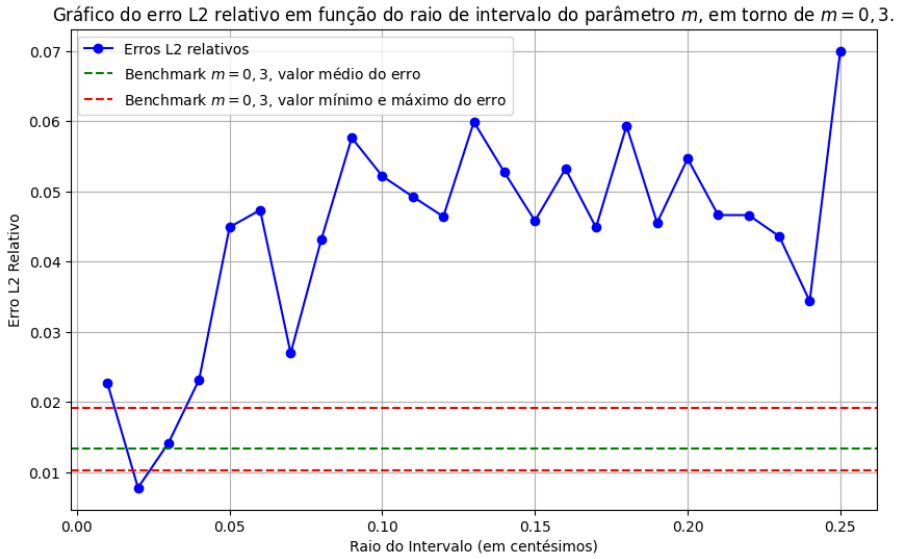
4.3.1.1 Resultados para variações do intervalo do parâmetro m

Com o *benchmark* estabelecido para o caso com o valor fixo de m, o estudo prosseguiu para testar a variação proposta, incluindo uma entrada extra na rede para representar o parâmetro m, sendo este variado a partir do intervalo $m \in [0{,}29,\ 0{,}31]$, descrito como raio de intervalo 0,01 centrado em $m = 0{,}3$, e finalizando no intervalo $m \in [0{,}05,\ 0{,}55]$, descrito como raio de intervalo 0,25 centrado em $m = 0{,}3$, sendo os testes variados em passos de 0,1 a cada iteração.

A evolução do erro L2 relativo pode ser observado na Figura 41, em que o resultado do *benchmark* foi adicionado para comparação.

Os testes contaram com treinamento utilizando o otimizador Adam, contando com o treinamento de 20 mil épocas para cada iteração de aumento de intervalo de m, em que foi utilizado um decaimento inversamente proporcional ao tempo para o fator de

Figura 41 – Evolução do erro L2 relativo em função do raio de intervalo em torno de m=0,3.



learning rate. O otimizador L-BFGS foi aplicado após 20 mil épocas, com critérios de parada baseados na convergência da perda, gradientes abaixo de 10^{-10} ou limite de 2000 iterações.

É observado, em média, uma característica de aumento do erro L2 relativo conforme o intervalo de m é aumentado, ao mesmo passo que não se observa um padrão conclusivo de evolução dos valores do erro L2, uma vez que os testes foram realizados com o otimizador Adam, um algoritmo que faz uso da descida do gradiente estocástico [29], partindo de inicializações aleatórias na rede, em que é esperado um resultado não determinístico a cada treinamento. Desta forma, seria necessária uma análise estatística mais robusta para uma conclusão mais assertiva da evolução dos erros devido o aumento do intervalo de parâmetro m. Porém, foram observados resultados positivos, uma vez que em relação ao teste de *benchmark* observaram-se resultados com erros na mesma ordem de grandeza, revelando uma possibilidade de se treinar redes neurais que representam famílias de soluções de equações diferenciais parciais não lineares.

4.3.1.2 Treinamento robusto a partir dos testes sistemáticos

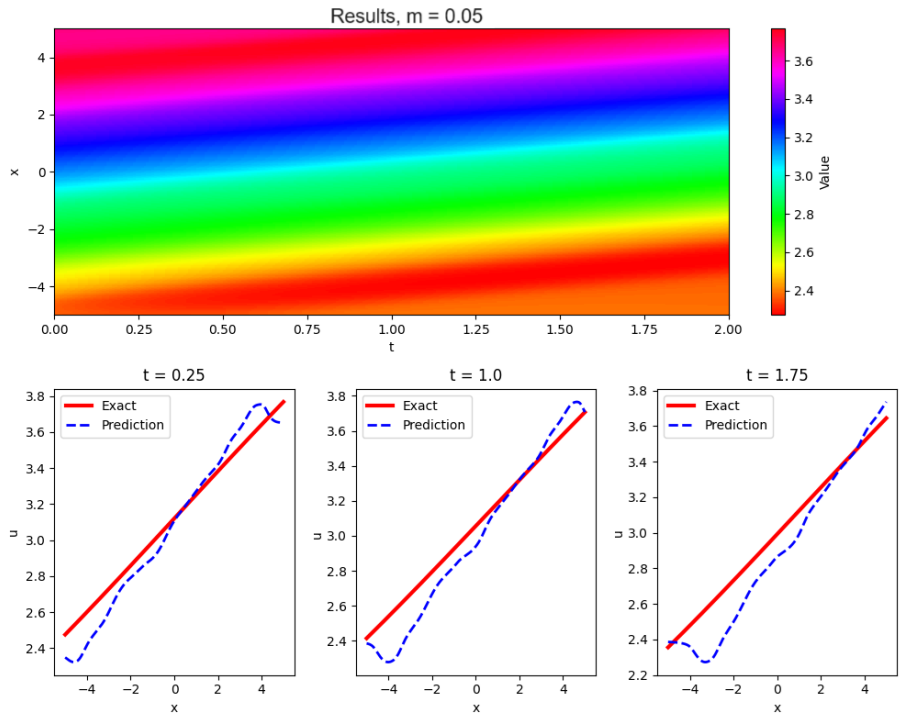
A partir dos resultados anteriores, em que não foi observada uma variação tão eminente dos valores de erro relativo L2 conforme a variação dos intervalos de m, foi testado em seguida um treinamento mais robusto da rede para um intervalo de variação maior do parâmetro m.

Foi utilizada a mesma quantidade de camadas e neurônios, porém ao invés de se treinar por 20 mil épocas, o modelo foi treinado por 100 mil épocas, com taxa de aprendizado constante, e em seguida por mais 100 mil épocas com taxa de aprendizado

reduzida gradativamente conforme o número de épocas. O resultado para o erro L2 relativo obtido foi $4{,}139 \times 10^{-2}$.

Uma vez que o modelo foi treinado, este se comporta como uma função com 3 variáveis. Uma avaliação da forma anterior, em que um *grid* contendo as coordenadas espaciais e temporais, e um mapa de cor indicando a predição referente ao par de coordenadas, deixa de ser possível. Porém, é possível avaliar o comportamento do modelo para cada valor de m ao se fixar o valor deste parâmetro. Para isso, deve-se plotar os resultados obtidos, de tal forma que foram escolhidos 5 valores dentro do intervalo, representados nas Figuras 42, 43, 44, 45 e 46.

Figura 42 – Resultados do modelo treinado, fixando $m = 0{,}05$, no domínio espaço temporal x e t.



É notado um aumento considerável do erro conforme o tempo se distancia da condição inicial em $t = 0$, assim como no geral as condições nas fronteiras $x = -5$ e $x = 5$, foram respeitadas. O modelo parece ter tido resultados menos convincentes para o caso em que m se aproxima de 0, mesmo o resultado em questão a princípio ser mais simples, se assemelhando a uma reta. Isso pode se dar como um indicativo da aproximação do caso em que o modelo passa a ser um resultado da equação de onda para o caso em que $m = 0$, ou ainda, simplesmente o fato do modelo apresentar características diferentes, semelhantes a uma reta, num contexto em que resultados se assemelham a arco tangentes, além de ser um resultado no limite do intervalo de m estudado.

Figura 43 – Resultados do modelo treinado, fixando $m = 0{,}175$, no domínio espaço temporal x e t.

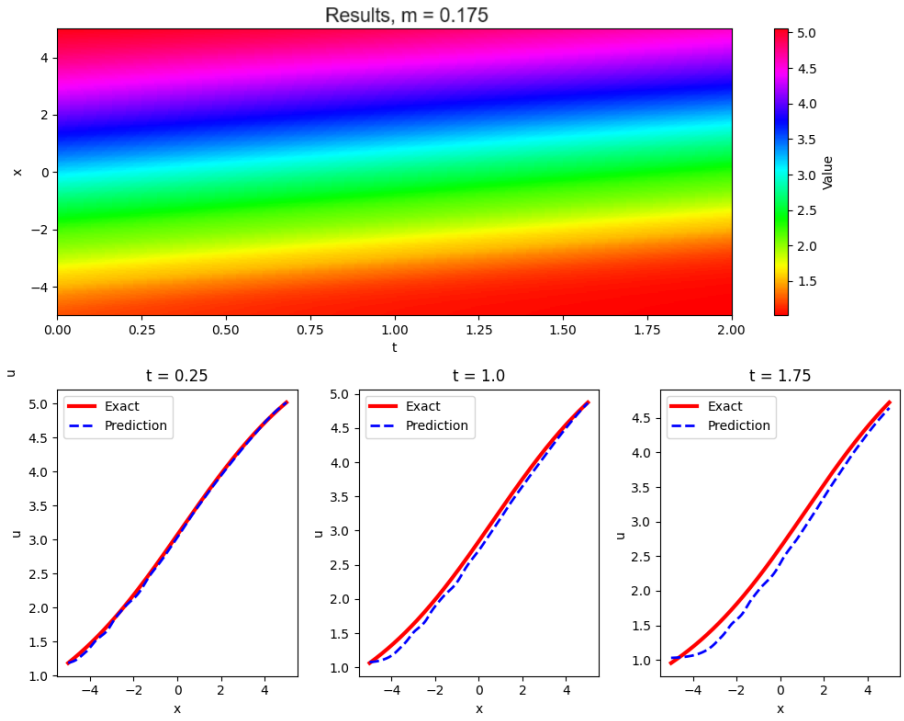


Figura 44 – Resultados do modelo treinado, fixando $m = 0{,}3$, no domínio espaço temporal x e t.

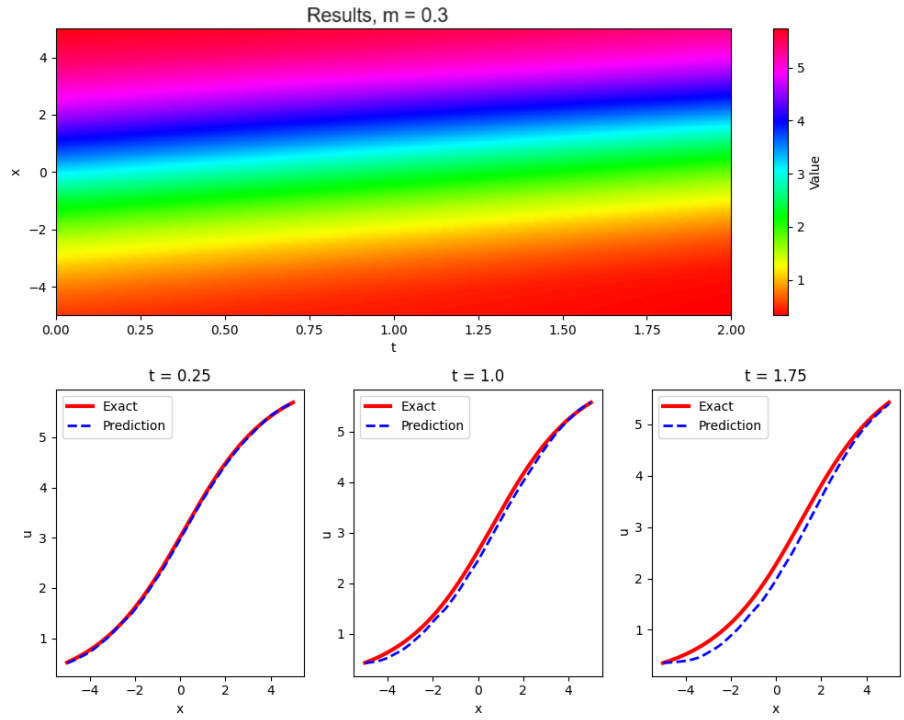


Figura 45 – Resultados do modelo treinado, fixando $m = 0{,}425$, no domínio espaço temporal x e t.

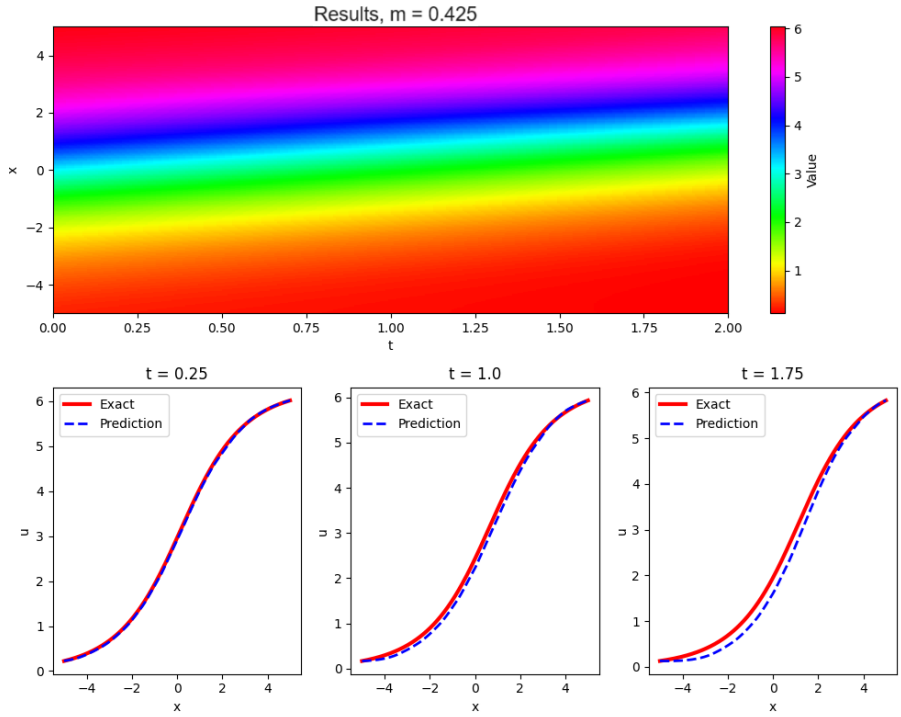
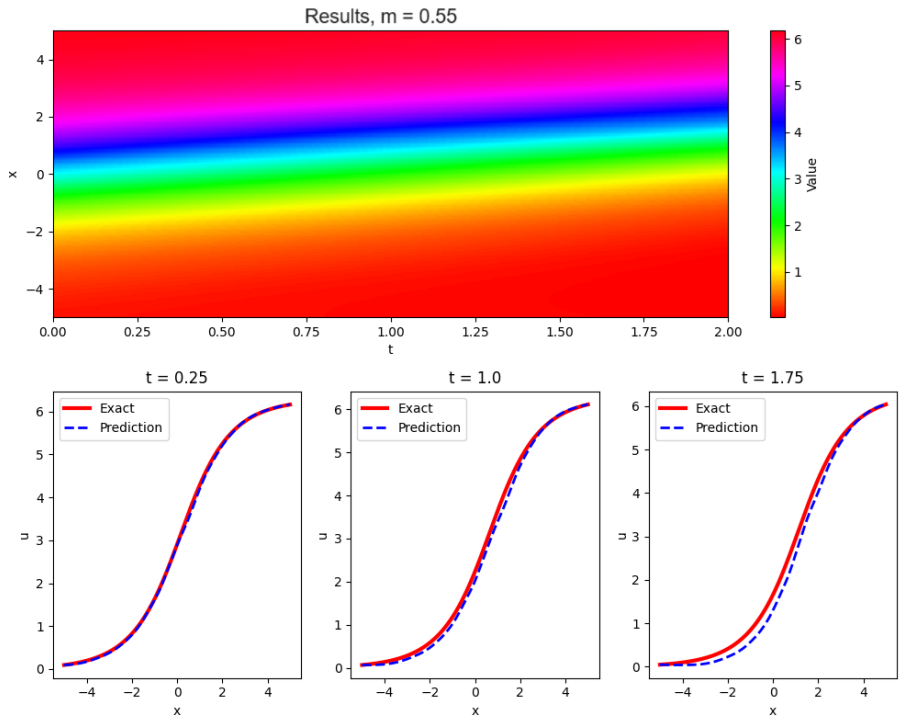


Figura 46 – Resultados do modelo treinado, fixando $m = 0{,}55$, no domínio espaço temporal x e t.



Ainda, é possível calcular o resultado dos erros referente a cada valor do parâmetro m. Calculando-se o erro L2 relativo para o domínio espaço temporal $\{(x,t) \mid x \in [-5,5], t \in [0,2]\}$, ao se fixar o valor do parâmetro m, o resultado para cada valor de m é representado na Figura 47.

Observou-se uma distribuição desigual dos erros, com um pico no limite inferior do intervalo de m, conforme descrito anteriormente, seguido de um mínimo para m próximo de 0,09. Os valores dos erros a partir de $m = 0{,}1$ aumentaram gradativamente conforme o valor do parâmetro m aumentou até próximo de $m = 0{,}25$. A partir do valor correspondente ao centro do intervalo, $m = 0{,}3$, os valores dos erros L2 relativos estabilizaram-se em aproximadamente 4,25%. Apesar da disparidade significativa entre os erros observados em relação ao parâmetro m, ela não foi excessiva, pois a ordem de grandeza dos erros manteve-se constante ao longo de todo o intervalo do parâmetro m testado.

Ainda em relação à distribuição de erros, é possível se observar os erros encontrados pelo modelo de forma semelhante ao estudado na seção anterior, onde ao se fixar algum valor de m é possível calcular o erro a partir da diferença entre o valor real e o valor predito. A título de exemplo, foram escolhidos valores para o parâmetro m, em torno do centro do intervalo do parâmetro m, sendo estes: $m = 0{,}175$ e $m = 0{,}425$, representando nas Figuras 48 e 49.

Figura 47 – Resultado do erro relativo L2 em relação ao parâmetro m, no domínio espaço temporal $\{(x,t) \mid x \in [-5,5], t \in [0,2]\}$

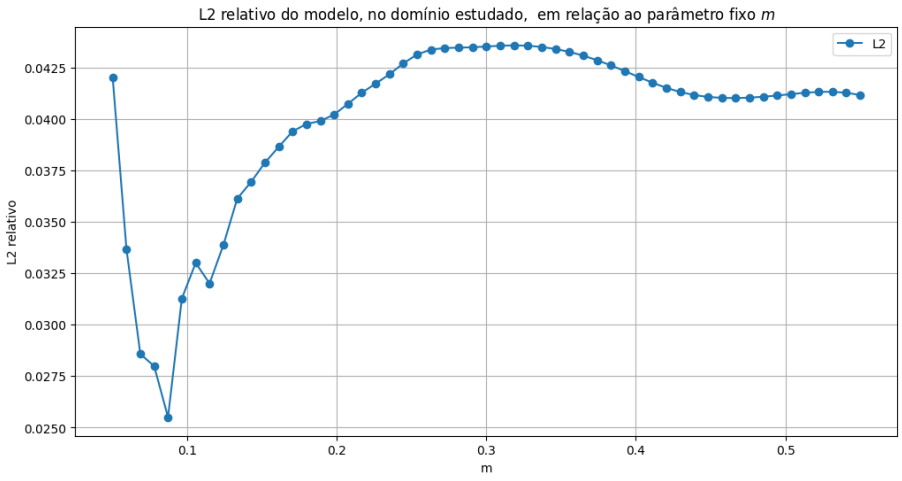
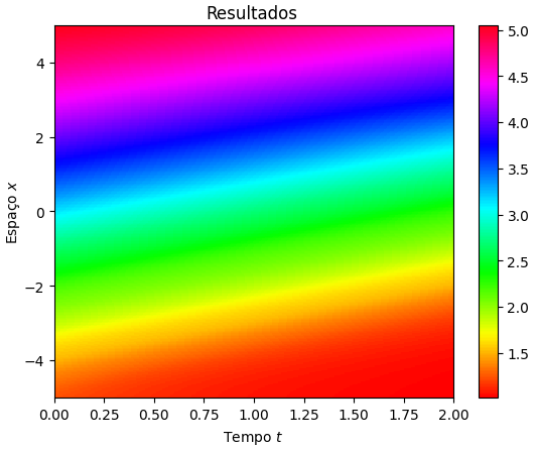
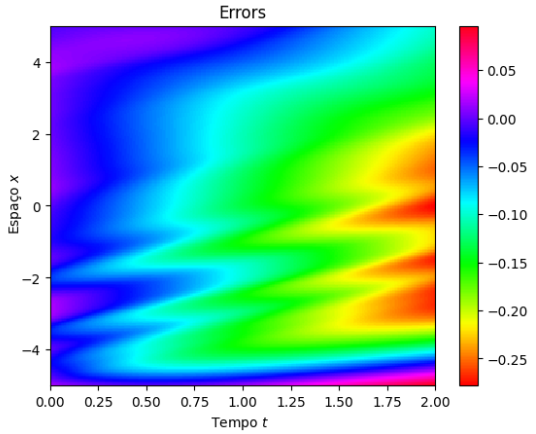


Figura 48 – Resultado do modelo da equação de Sine-Gordon com o parâmetro m fixado em 0,175, treinado por 200 mil épocas para o intervalo $m \in [0{,}05,\ 0{,}55]$, e seus respectivos erros.

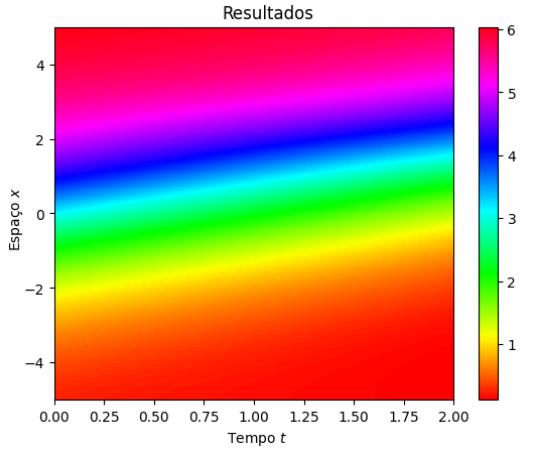


(a) Resultados do modelo para $m = 0{,}175$.

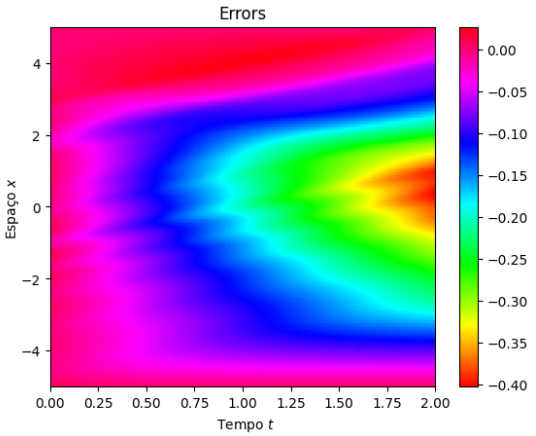


(b) Distribuição dos erros para $m = 0{,}175$.

Figura 49 – Resultado do modelo da equação de Sine-Gordon com o parâmetro m fixado em 0,425, treinado por 200 mil épocas para o intervalo $m \in [0{,}05,\ 0{,}55]$, e seus respectivos erros.



(a) Resultados do modelo para $m = 0{,}425$.



(b) Distribuição dos erros para $m = 0{,}425$.

É notado nas Figuras 48 e 49, um aumento dos erros conforme se distancia das condições iniciais fornecidas para o problema, em $t = 0$, assim como das condições de contorno para $x = -5$, e $x = 5$. Pode-se observar que a borda direita do modelo apresenta maior presença dos erros para o problema.

Conclui-se que, dados os erros observados, o modelo representou a equação Sine-Gordon, sendo a solução referente não somente a um valor específico para o parâmetro m, mas sim a um intervalo, representando uma solução contínua, independente de malhas, tanto para os valores de coordenadas x e t, no domínio espaço temporal, tanto quanto para valores de m dentro do intervalo estudado.

Foi testado em seguida a possibilidade de se minimizar os erros encontrados, com o uso de uma condição de contorno extra.

4.3.2 Resultados utilizando uma condição de contorno extra

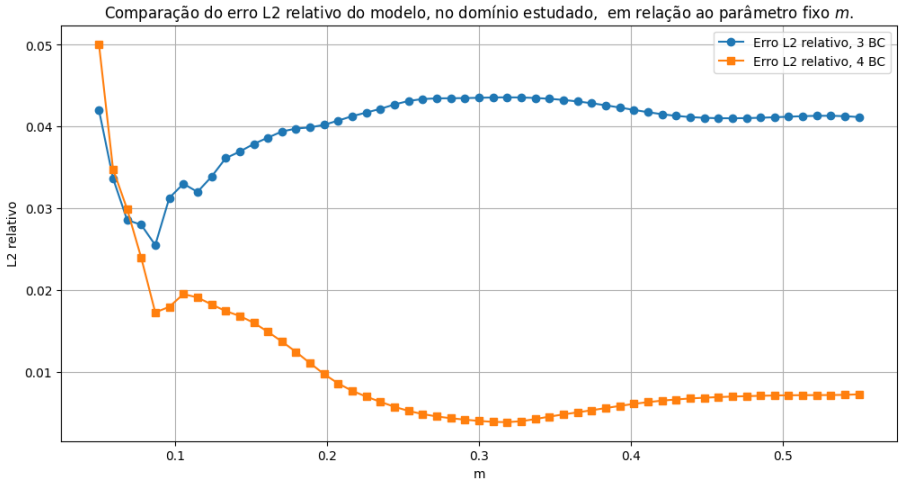
Pode ser observado na distribuição dos erros da seção anterior, que conforme a solução se distanciou da condição inicial fornecida à máquina, houve uma piora significativa nos resultados. Juntamente com as condições de contorno em $x = -5$ e $x = 5$, apenas a fronteira direita do domínio da solução não possuía vínculos definidos por condições de contorno, o que resulta em uma distribuição de erros significativamente maior em comparação às demais bordas.

Com o objetivo de se estudar como a solução se comportaria ao se contornar este problema, fornecendo uma condição de contorno para $t = 2$, de forma que as quatro fronteiras do problema fossem submetidas a vínculos físicos, uma condição de contorno extra foi implementada, da forma $u(x, m, t = 2) = f(x, m)$, ou seja, foram fornecidos dados referentes às imposições físicas conhecidas da solução do problema em $t = 2$, totalizando uma quarta condição de contorno do modelo.

4.3.2.1 Comparação com o resultado com três condições de contorno

Foi utilizado para avaliação e comparação, o caso relativo ao resultado anterior, para m centrado em 0,3, compreendendo o intervalo $m \in [0{,}05,\ 0{,}55]$. Desta forma, o erro L2 relativo referente ao domínio $\{(x, t) \mid x \in [-5,\ 5],\ t \in [0, 2]\}$ foi calculado, ao se fixar o valor do parâmetro m no modelo. Os resultados, comparados com o caso de 3 condições de contorno, estudados anteriormente, foram representados na Figura 50.

Figura 50 – Comparação do erro L2 relativo, nos casos de 3 e 4 condições de contorno (3 BC e 4 BC, respectivamente), calculados no domínio espaço temporal $\{(x, t) \mid x \in [-5, 5],\ t \in [0, 2]\}$, em relação ao parâmetro fixo m.



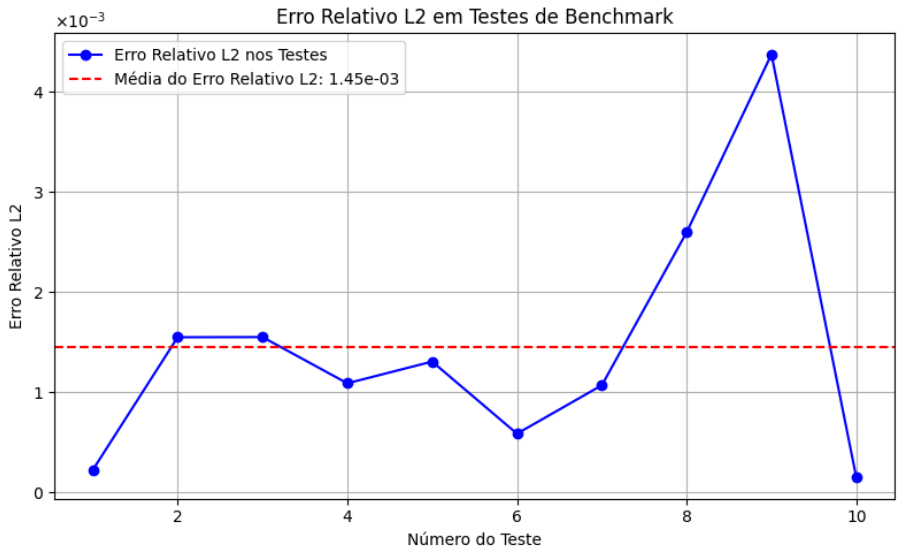
É observada uma diferença significativa nos resultados, sendo que a característica do modelo apresentar resultados piores para valores próximos de 0, para o parâmetro m foi mantida. Porém, observando valores a partir de 0,1 foi notada uma significativa melhora dos resultados, mostrando um impacto significativo nos resultados do modelo ao se informar vínculos físicos extras para o modelo.

4.3.2.2 Avaliação do modelo com 4 condições de contorno em intervalos ampliados do parâmetro m

Após verificar que os resultados para m centrado em 0,3 apresentaram desempenho superior com a inclusão da quarta condição de contorno, foi realizado um novo teste considerando $m = 0{,}5$ como valor central. Nesse teste, utilizou-se um intervalo maior para o parâmetro m, avaliando a capacidade do modelo em generalizar soluções para um espectro ampliado, mantendo as 4 condições de contorno.

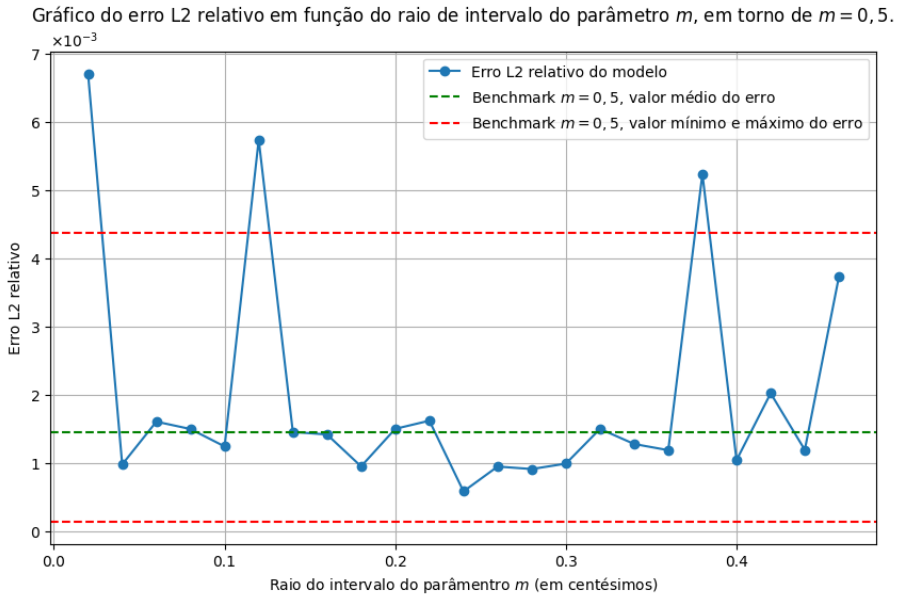
Com a quarta condição de contorno implementada em $t = 2$, o modelo foi treinado de forma similar ao caso com 3 condições de contorno, agora com m centrado em 0,5. Foi realizado um *benchmark* específico para esse valor, utilizando uma rede com duas entradas. Seguindo o procedimento anterior, devido à natureza estocástica do otimizador *Adam*, foram realizados 10 treinamentos para calcular o erro relativo L_2. Os resultados, apresentados na Fig. 51, indicaram uma média do erro L_2 de $1{,}447 \times 10^{-3}$.

Figura 51 – Resultados do modelo da equação de Sine-Gordon com o parâmetro m fixado em 0,5 treinado por 20 mil épocas.



Em seguida, foi dado início a uma bateria de testes para o modelo com 4 condições de contorno, em que foram testados diferentes intervalos para o parâmetro m, partindo-se do intervalo $m \in [0{,}48, 0{,}52]$, descrito como raio de intervalo 0,02 em torno de $m = 0{,}5$. Os testes foram realizados de forma que o raio de intervalo da variável m tivesse incrementos com variações de de 0,02 a cada passo, finalizando no raio de intervalo 0,46,

Figura 52 – Evolução do erro L2 relativo em função do raio de intervalo em torno de $m = 0{,}5$. Caso do modelo que conta com 4 condições de contorno para a equação de Sine-Gordon com parâmetro m variável.



correspondendo ao intervalo $m \in [0{,}04, 0{,}96]$ sendo estes resultados representados na Figura 52. O teste de cada intervalo contou com 20 mil épocas de treinamento, utilizando o otimizador Adam.

Nota-se que os valores de erros correspondentes aos intervalos ficaram em sua maioria próximos ao erro médio presente no caso de Benchmark. Nota-se também que os mínimos dos erros não foram menores que o caso de *Benchmark*, e os máximos ultrapassaram os maiores erros do teste comparativo, o que era esperado. O resultado indica uma congruência melhor com o caso de *Benchmark*, que possui o parâmetro m fixo, para um modelo com o parâmetro m variado, num caso em que mais condições de contorno foram utilizadas no treinamento.

Em seguida o modelo foi testado num caso de um treinamento mais robusto para o caso de maior intervalo de m estudado.

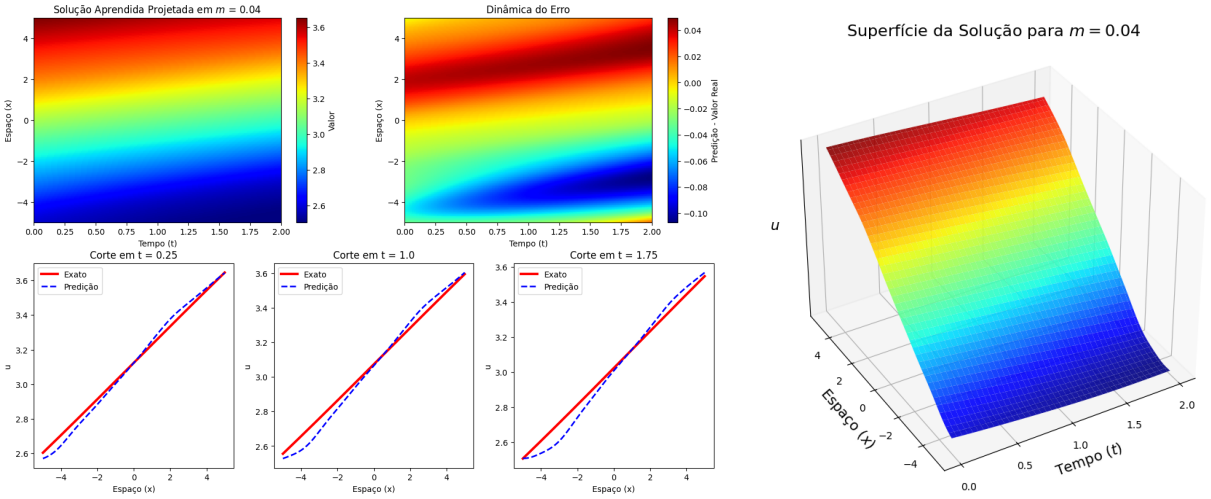
4.3.2.3 Treinamento robusto do modelo com 4 condições de contorno

Foi mantida a mesma arquitetura, constituída em: `[3] + [200] * 6 + [1]`, com 3 camadas de entrada representando as 2 coordenadas espaço-temporais (x, t), sendo o domínio espaço-temporal o mesmo do teste anterior, e o parâmetro $m \in [0{,}04, 0{,}96]$, seguidos por 6 camadas ocultas de 200 neurônios cada, e uma camada de saída que representa o resultado da função que depende das três entradas: $u(x, m, t)$. O treinamento foi feito de forma mais robusta, e após alguns testes o melhor resultado encontrado foi com o treinamento por 30 mil épocas, com o treinamento das 20 mil épocas finais sendo feito com valor reduzido de taxa de aprendizado. O resultado para o erro L2 relativo obtido

foi $2{,}809 \times 10^{-3}$.

De forma semelhante ao caso anterior, uma vez que o modelo foi treinado, este se comporta como uma função com 3 variáveis, uma avaliação da forma de um *grid* contendo as coordenadas espaciais e temporais, e um mapa de cor indicando a predição referente ao par de coordenadas, deixa de ser possível. Porém é possível avaliar o comportamento do modelo para cada valor de m, ao se fixar o valor deste parâmetro. Foi feito o gráfico dos resultados obtidos, sendo 5 possíveis valores escolhidos dentro do intervalo representado nas Figuras 53, 54, 55, 56 e 57.

Figura 53 – Resultados do modelo treinado com $m = 0.04$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida.



(a) (b)

Figura 54 – Resultados do modelo treinado com $m = 0.27$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida.

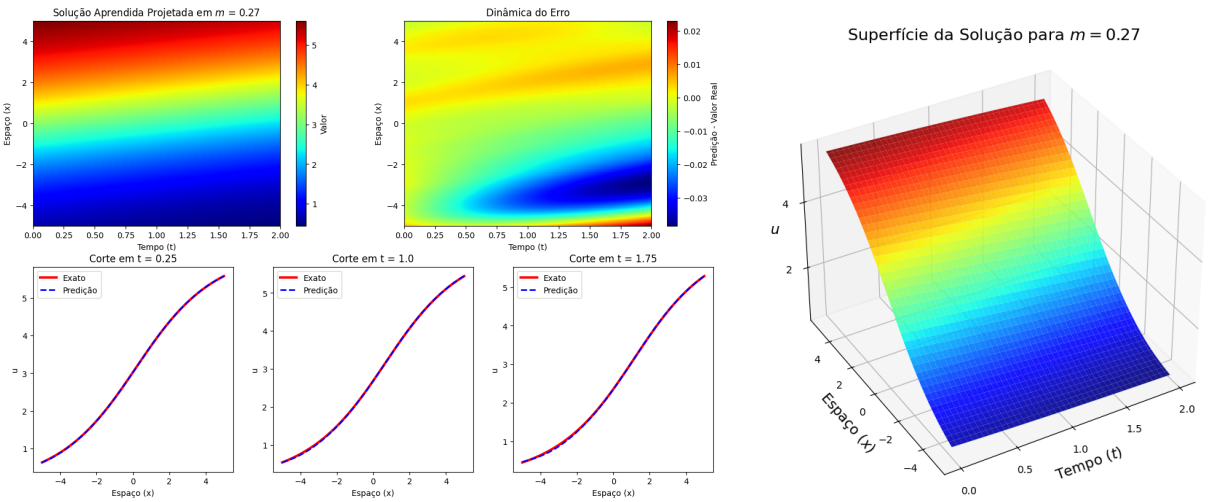


Figura 55 – Resultados do modelo treinado com $m = 0.50$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida.

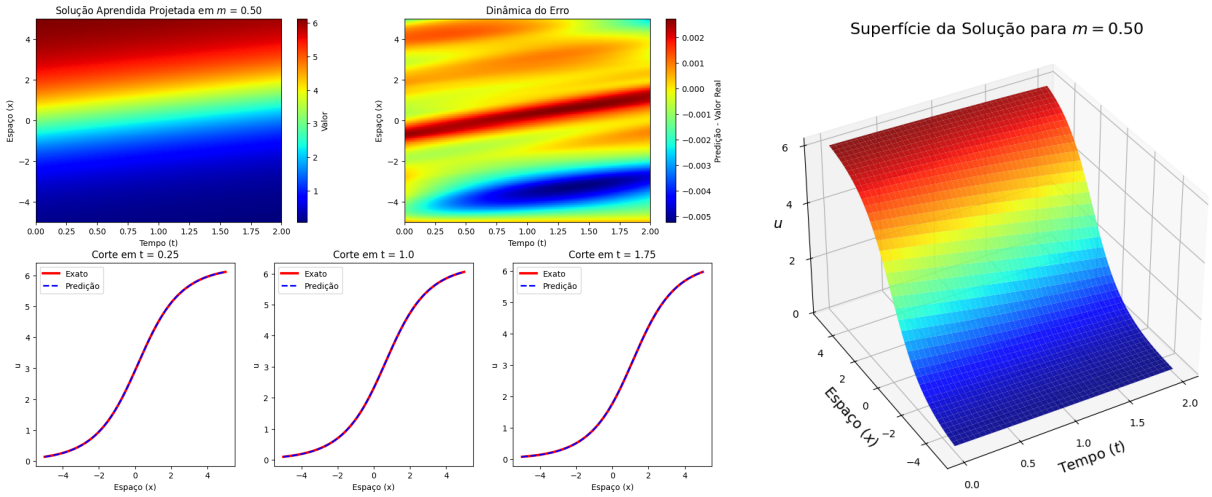


Figura 56 – Resultados do modelo treinado com $m = 0.73$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida.

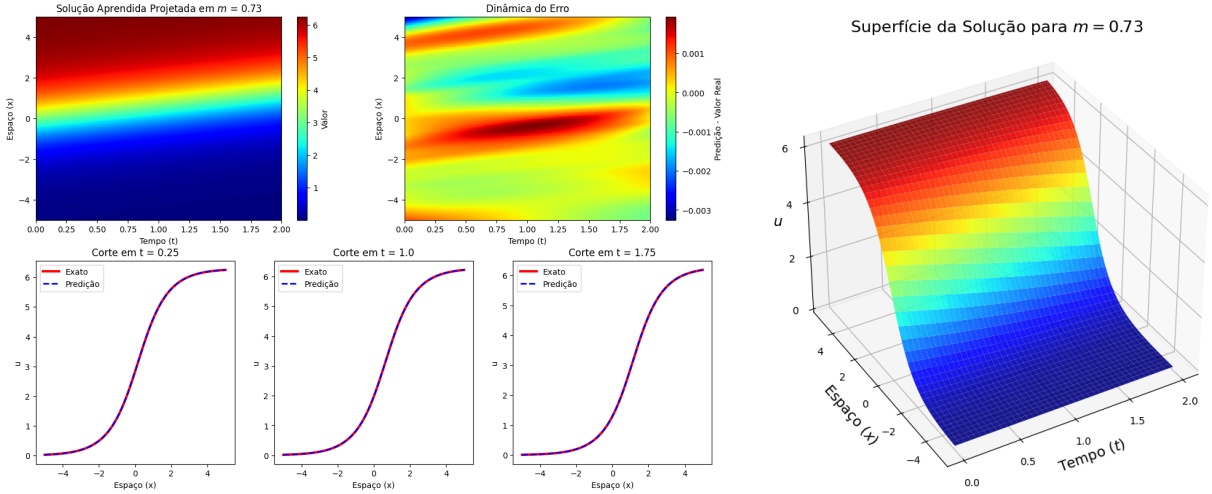
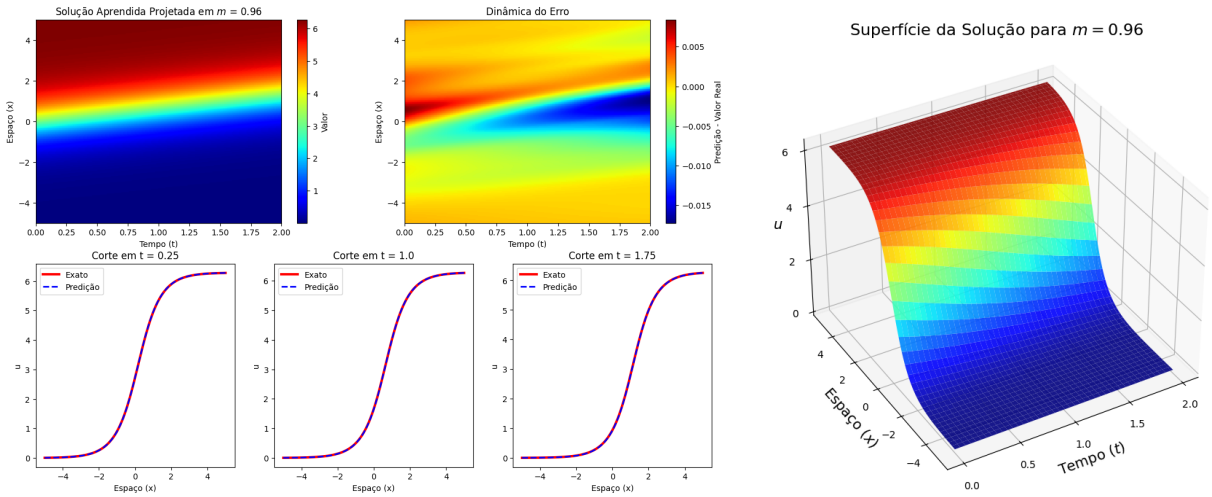


Figura 57 – Resultados do modelo treinado com $m = 0.96$. (a) Solução no domínio espaço-temporal x e t. (b) Gráfico de superfície 3D da solução aprendida.



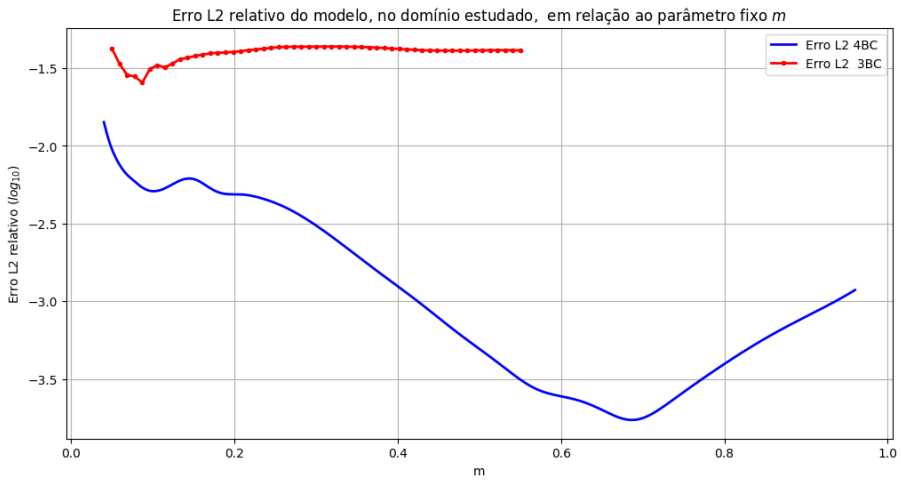
É notada a influência positiva de se adicionar a nova condição de contorno, onde os resultados para tempos próximos de 2 se demonstram com erros a princípio não perceptíveis, exceto no caso de $m = 0{,}04$. De forma semelhante ao modelo com 3 condições de contorno, foi notado uma piora dos resultados para m próximos de 0, por motivos semelhantes aos citados anteriormente, em que o modelo está na borda do intervalo fornecido. Nesta condição, o resultado se aparenta mais a uma reta, e também, ao se aproximar de 0, o modelo se aproxima do modelo da equação de onda, onde uma generalização pode não ser aplicável trivialmente.

Os erros relacionados às predições do modelo podem ser melhor visualizados no gráfico de dinâmica dos erros, que representa a diferença entre o valor previsto e o valor real do problema. Pode ser notado que não há mais um padrão definido de aumento dos erros nas bordas da direita. Também podem ser observados os valores de diferença entre o valor previsto e o valor real significativamente menores em comparação com os resultados referentes ao modelo com 3 condições de contorno.

Como o modelo se comporta como uma função contínua, é possível plotar os valores de erros obtidos para qualquer valor de m, de forma similar à seção anterior. Sendo assim, foi calculado o valor do erro L2 relativo referente ao domínio espaço temporal $\{(x,t) \mid x \in [-5,\,5], t \in [0,\,2]\}$, ao fixar valores de m, o que foi representado na Figura 58. Frisando a capacidade de predições contínuas do modelo, optou-se por adicionar uma resolução maior de pontos para este resultado. Para fins de comparação, também foi realizado o cálculo contínuo de erros em relação ao domínio de m estudado, incluindo o resultado anterior do modelo com 3 condições de contorno.

Foi optada uma notação log_{10} no eixo y, referente aos erros, uma vez que o modelo obteve resultados com contrastes de erros para o intervalo do modelo, para valores pró-

Figura 58 – Resultado do erro relativo L2 em relação ao parâmetro m, no domínio espaço temporal $\{(x,t) \mid x \in [-5,5],\ t \in [0,2]\}$, para o modelo com 4 condições de contorno.



ximos de 0 de m, como comentado anteriormente. Porém, é observado que o máximo do erro do modelo com 4 condições de contorno, em $m = 0.04$, L2 $= 1{,}358 \times 10^{-2}$, possui um valor inferior ao mínimo do erro do modelo com 3 condições de contorno, em $m = 0{,}087$, com valor de erro L2 $= 2{,}799 \times 10^{-2}$, que foi adicionado para conferência.

4.3.3 Análise do Tempo de Treinamento e Impacto da Dimensão Extra de Entrada

O tempo de treinamento tanto para o caso de referência com parâmetros fixos quanto para o modelo de PINNs generalizado com valores variáveis de parâmetros foi em torno de 280 segundos, representando tempo hábil. Essa semelhança na duração do treinamento pode ser explicada pela forma como o número de parâmetros treináveis muda ao adicionar uma dimensão extra de entrada, como o parâmetro m.

Em uma rede neural totalmente conectada, o número de parâmetros treináveis está diretamente relacionado às conexões entre os neurônios das camadas. Quando uma dimensão extra é adicionada à entrada, isso afeta apenas o número de pesos associados às conexões entre a camada de entrada e a primeira camada oculta, pois são necessários pesos adicionais para conectar a nova dimensão a todos os neurônios dessa camada oculta. No entanto, como o restante da rede permanece inalterado, o impacto total é mínimo. Em modelos típicos de redes neurais, que possuem centenas de milhares de parâmetros, essa adição corresponde a apenas algumas centenas de novos pesos, o que é insignificante em relação ao total.

Como resultado, expandir o espaço de entrada para incluir o parâmetro variável m permite que o modelo de PINNs com variação de intervalos generalize em uma gama de condições sem impacto significativo no tempo de treinamento, demonstrando uma escalabilidade eficiente no tratamento de EDPs parametrizadas.

O tempo de treinamento observado de 280 segundos pode ser considerado notavelmente baixo, especialmente quando se leva em conta a duração de muitos projetos acadêmicos envolvendo redes neurais complexas. De fato, trabalhos de pesquisa frequentemente reportam tempos de treinamento que se estendem por dias ou semanas, mesmo quando otimizados para os recursos computacionais tipicamente disponíveis em universidades. Por exemplo, [60] discutem o pré-treinamento de modelos de linguagem na casa de 1 bilhão de parâmetros, como o Pythia-1B, que, mesmo em um cenário acadêmico com otimizações, pode demandar cerca de 18 dias em um conjunto de 4 GPUs A100. Essa perspectiva ilustra que um período de 280 segundos realmente indica um espaço considerável para customização e desenvolvimento de um modelo de predição mais robusto, dado que muitos empreendimentos acadêmicos já partem de uma escala de tempo muito superior.

5 Conclusão

Os resultados apresentados neste trabalho demonstram o grande potencial das Redes Neurais Informadas por Física (PINNs) como uma ferramenta robusta para a resolução de equações diferenciais, especialmente em cenários não lineares envolvendo sólitons e parâmetros variáveis. A investigação foi estruturada em três focos principais, cada um abordando desafios específicos e contribuindo para um entendimento mais profundo da aplicação de redes neurais informadas por física em problemas complexos.

No primeiro foco, a aplicação de PINNs a uma equação diferencial ordinária de segunda ordem para um intervalo de variação de parâmetros, evidenciou a capacidade do modelo em aprender padrões complexos e generalizar soluções para diferentes intervalos de parâmetros. Os erros $L2$ médios, na faixa de 10^{-3} a $10^{-3.5}$, indicaram que a escolha de intervalos de parâmetros adequados é essencial para o desempenho do modelo, uma vez que intervalos maiores de variação de parâmetros demonstraram uma performance melhor do que um com menos variação, dependendo do intervalo escolhido.

O segundo foco explorou a aplicação de PINNs à equação de Korteweg-de Vries, investigando sua capacidade de modelar sólitons isolados e interações entre dois sólitons. A implementação com a biblioteca DeepXDE mostrou-se eficiente para equações não lineares, permitindo um entendimento mais profundo sobre a estrutura necessária para modelagens complexas.

Os resultados demonstraram a eficácia do modelo em simular fenômenos não lineares com um número reduzido de dados. Utilizando apenas 100 pontos relacionados exclusivamente às condições iniciais, o modelo foi capaz de fornecer uma solução contínua, apresentando resultados fisicamente coerentes dentro do domínio estudado. Para um sóliton, o erro relativo foi de 3.76×10^{-4}, indicando alta precisão na modelagem do fenômeno isolado. Já para o caso da colisão de dois sólitons, o erro relativo foi de $1{,}53 \times 10^{-2}$, um valor relativamente mais alto. Ainda assim, mostrou-se suficiente para que o fenômeno da colisão entre sólitons da equação de KdV fosse corretamente representado, mesmo com apenas 100 pontos de dados provenientes de uma única condição de contorno imposta em $t = 0$, o que é notável considerando a complexidade do cenário de interação.

A análise da arquitetura mostrou, para os dados utilizados, que redes mais profundas (8 camadas) são adequadas para sólitons isolados, enquanto arquiteturas mais simples (2 camadas) são eficazes para colisões. Ainda, foi observada uma arquitetura mínima de duas camadas para a distinção de dois sólitons no domínio estudado, indicando a importância da escolha de hiperparâmetros no desempenho do modelo.

Além disso, a implementação de vínculos físicos foi realizada de forma simples

e direta com o uso do DeepXDE, destacando a flexibilidade da biblioteca na inclusão de condições físicas conhecidas no problema. No entanto, esses vínculos não trouxeram ganhos significativos na precisão, sugerindo que a equação diferencial e as condições iniciais já impõem restrições suficientes para o caso estudado.

Dessa forma, a investigação contribuiu para um melhor entendimento da modelagem não linear de sólitons na equação de KdV por redes neurais, ao indicar a importância de um estudo sobre hiperparâmetros da rede neural. Ao mesmo tempo em que demonstrou o potencial e as limitações das PINNs nesse contexto, alinhando-se aos objetivos estabelecidos para essa etapa do estudo.

O terceiro foco apresentou o modelo mais abrangente e com resultados mais significativos, ao aplicar uma extensão das PINNs à equação de Sine-Gordon. Esse enfoque consolidou a tarefa de generalização explorada no primeiro foco, agora em um contexto não linear, conforme investigado no segundo foco. A extensão proposta permitiu resolver a equação parametrizada, generalizando as soluções ao longo de um intervalo específico do parâmetro m. Ao incorporar uma dimensão adicional para esse parâmetro variável, o modelo generalizou as soluções sem a necessidade de retreinamento para cada novo valor de m. Além disso, o tempo de treinamento permaneceu estável (cerca de 280 segundos), mesmo com a dimensão extra, evidenciando a eficiência computacional da abordagem.

A inclusão de condições de contorno adicionais em $t = 2$ aprimorou a precisão do modelo, especialmente em regiões críticas do domínio. Foram observados erros na ordem de grandeza de 10^{-2} com 3 condições de contorno e 10^{-3} com 4 condições de contorno, demonstrando o impacto positivo de informações adicionais nas bordas do domínio. No entanto, desafios persistiram para valores de m próximos de zero, onde a solução se aproxima da forma linear da equação de onda, indicando uma possível limitação na capacidade de generalização do modelo nesse regime.

Os resultados obtidos confirmam a hipótese inicial de que redes neurais podem ser adaptadas para abordar múltiplos cenários físicos com um custo computacional incremental reduzido. A escolha da equação de Sine-Gordon, um problema menos explorado no contexto de redes neurais, expandiu o escopo metodológico, enquanto a comparação sistemática entre modelos treinados com parâmetros fixos e modelos generalizados validou a eficiência da abordagem proposta. Isso pode fornecer contribuições importantes para a otimização computacional da modelagem de fenômenos não lineares.

Como direções futuras, sugere-se a exploração de arquiteturas adaptativas, capazes de ajustar dinamicamente sua complexidade para lidar com regiões problemáticas, como $m \to 0$. Além disso, a subdivisão do espaço de parâmetros com modelos especializados pode oferecer ganhos adicionais de precisão. A aplicação dessa abordagem a equações diferenciais parciais com múltiplos parâmetros variáveis também representa um caminho promissor a ser explorado em estudos futuros.

Em síntese, este trabalho não apenas validou a versatilidade das PINNs em cenários desafiadores, mas também abriu novos caminhos para sua aplicação em sistemas físicos mais complexos, nos quais a generalização e a eficiência computacional são fatores essenciais. A integração de conhecimentos físicos com a flexibilidade das redes neurais fortalece o potencial dessa abordagem como uma ponte entre modelagem teórica e soluções computacionais inovadoras, oferecendo novos paradigmas para a resolução de equações diferenciais em contextos variados.

Referências

1 MAZIAR, R.; PERDIKARIS, P.; KARNIADAKIS, G. e. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, v. 378, p. 686–707, 2019. Citado 13 vezes nas páginas 19, 29, 31, 32, 33, 34, 53, 54, 55, 68, 69, 71 e 85.

2 CUOMO, S. et al. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, Springer Science and Business Media LLC, v. 92, n. 3, jul. 2022. Disponível em: <https://doi.org/10.1007/s10915-022-01939-z>. Citado na página 19.

3 LOGAN, J. D. *An introduction to nonlinear partial differential equations*. 2. ed. Hoboken, NJ: Wiley-Blackwell, 2008. (Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts). Citado na página 19.

4 MANUKURE, S.; BOOKER, T. A short overview of solitons and applications. *Partial Differential Equations in Applied Mathematics*, v. 4, p. 100140, 2021. ISSN 2666-8181. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666818121000747>. Citado na página 19.

5 ABLOWITZ, M.; CLARKSON, P. *Solitons, Nonlinear Evolution Equations and Inverse Scattering*. 1. ed. Cambridge: Cambridge University Press, 1991. Citado 6 vezes nas páginas 19, 45, 46, 49, 59 e 64.

6 CUEVAS-MARAVER, J.; KEVREKIDIS, P. G.; WILLIAMS, F. (Ed.). *The sine-Gordon Model and its Applications: From Pendula and Josephson Junctions to Gravity and High-Energy Physics*. Springer International Publishing, 2014. ISSN 2196-0003. ISBN 9783319067223. Disponível em: <http://dx.doi.org/10.1007/978-3-319-06722-3>. Citado 2 vezes nas páginas 19 e 50.

7 LI, J.; CHEN, J.; LI, B. Gradient-optimized physics-informed neural networks (gopinns): a deep learning method for solving the complex modified kdv equation. *Nonlinear Dynamics*, v. 107, n. 1, p. 781–792, 2022. Cited By 15. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85118662459&doi=10.1007%2fs11071-021-06996-x&partnerID=40&md5=2b3cfe0fc961b398b18be62ed6e406ce>. Citado na página 19.

8 XIAO, C. et al. Physics-informed neural network for solving coupled korteweg-de vries equations. In: . [s.n.], 2021. v. 2031, n. 1. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85117617561&doi=10.1088%2f1742-6596%2f2031%2f1%2f012056&partnerID=40&md5=1a1ab5cd6cf80a7841dc5bfdf48121d1>. Citado na página 19.

9 ZHU, X.; HE, M.; SUN, P. Comparative studies on mesh-free deep neural network approach versus finite element method for solving coupled nonlinear hyperbolic/wave equations. *International Journal of Numerical Analysis and Modeling*, v. 19, n. 5, p. 603–629, 2022. Cited By 1. Disponível em: <https://www.scopus.com/inward/record.uri?eid=

2-s2.0-85135999902&partnerID=40&md5=0472300b6dc38c931f5d0a9a2ff4ff1e>. Citado na página 19.

10 LIN, S.; CHEN, Y. Physics-informed neural network methods based on miura transformations and discovery of new localized wave solutions. *Physica D: Nonlinear Phenomena*, v. 445, 2023. Cited By 2. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85145021675&doi=10.1016%2fj.physd.2022.133629&partnerID=40&md5=ba40f3bbab221dc28eead56ab326911f>. Citado na página 19.

11 JAGTAP, A.; KHARAZMI, E.; KARNIADAKIS, G. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, v. 365, 2020. Cited By 199. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083000605&doi=10.1016%2fj.cma.2020.113028&partnerID=40&md5=0086485fb6e0387ff31ea3921ae25351>. Citado 2 vezes nas páginas 19 e 20.

12 WEN, Y.; CHAOLU, T. Learning the nonlinear solitary wave solution of the korteweg–de vries equation with novel neural network algorithm. *Entropy*, v. 25, n. 5, 2023. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85160527877&doi=10.3390%2fe25050704&partnerID=40&md5=b129a7bbc92ea68b182563037aacf53d>. Citado na página 19.

13 ZHANG, H. et al. Enforcing generalized conditional symmetry in physics-informed neural network for solving the kdv-like equation with robin initial/boundary conditions. *Nonlinear Dynamics*, v. 111, n. 11, p. 10381–10392, 2023. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85150071795&doi=10.1007%2fs11071-023-08361-6&partnerID=40&md5=9c3cd77867806e0be292da68542669c3>. Citado na página 19.

14 ZHANG, Y. et al. The new simulation of quasiperiodic wave, periodic wave, and soliton solutions of the kdv-mkdv equation via a deep learning method. *Computational Intelligence and Neuroscience*, v. 2021, 2021. Cited By 3. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85120820729&doi=10.1155%2f2021%2f8548482&partnerID=40&md5=64b1730300b4ae54b9ff94d48ec6b9c3>. Citado na página 19.

15 LI, J.; CHEN, Y. A deep learning method for solving third-order nonlinear evolution equations. *Communications in Theoretical Physics*, v. 72, n. 11, 2020. Cited By 28. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85094957699&doi=10.1088%2f1572-9494%2fabb7c8&partnerID=40&md5=c6980751201bb221456200da5ddc298e>. Citado na página 19.

16 GUO, Y. et al. Solving partial differential equations using deep learning and physical constraints. *Applied Sciences (Switzerland)*, v. 10, n. 17, 2020. Cited By 36. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85090205179&doi=10.3390%2fapp10175917&partnerID=40&md5=1c78bf6b16c5b5bf3240271375bc0d50>. Citado na página 19.

17 CHEN, Y.; YAN, J.; ZHONG, X. Cell-average based neural network method for third order and fifth order kdv type equations. *Frontiers in Applied Mathematics and Statistics*, v. 8, 2022. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85142233531&doi=10.3389%2ffams.2022.1021069&partnerID=40&md5=ca0c81a1299f098acf29244cebeee126>. Citado na página 19.

18 ZHU, J.; CHEN, Y. Data-driven solutions and parameter discovery of the nonlocal mkdv equation via deep learning method. *Nonlinear Dynamics*, v. 111, n. 9, p. 8397–8417, 2023. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85147358590&doi=10.1007%2fs11071-023-08287-z&partnerID=40&md5=3c00102368d33e7cb55fad446534ca02>. Citado na página 19.

19 LI, J.; CHEN, Y. A physics-constrained deep residual network for solving the sine-gordon equation. *Communications in Theoretical Physics*, v. 73, n. 1, 2021. Cited By 14. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85098550903&doi=10.1088%2f1572-9494%2fabc3ad&partnerID=40&md5=32edf6bef7813f83511ca1d47042889b>. Citado na página 19.

20 ZHOU, Z.; WANG, L.; YAN, Z. Data-driven discoveries of bäcklund transformations and soliton evolution equations via deep neural network learning schemes. *Physics Letters, Section A: General, Atomic and Solid State Physics*, v. 450, 2022. Cited By 1. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85136542999&doi=10.1016%2fj.physleta.2022.128373&partnerID=40&md5=aaf86bc3a3fd23b86ee182d1febee77f>. Citado na página 19.

21 CHOLLET, F. *Deep Learning with Python*. 2. ed. Shelter Island, NY: Manning Publications Co., 2021. Citado 6 vezes nas páginas 22, 24, 25, 26, 27 e 28.

22 BENKO, A.; LÁNYI, C. S. History of artificial intelligence. Information Science Reference, Hershey, NY, p. 1759–1761, 2009. Citado na página 22.

23 MINSKY, M.; PAPERT, S. Perceptrons - an introduction to computational geometry. 1969. Citado na página 23.

24 HAENLEIN, M.; KAPLAN, A. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, v. 61, n. 4, p. 5–14, 2019. Disponível em: <https://doi.org/10.1177/0008125619864925>. Citado na página 23.

25 LU, L. et al. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, Society for Industrial & Applied Mathematics (SIAM), v. 63, n. 1, p. 208–228, jan 2021. Disponível em: <https://doi.org/10.1137%2F19m1274067>. Citado 6 vezes nas páginas 29, 30, 31, 34, 53 e 54.

26 BAYDIN, A. et al. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, v. 18, p. 1–43, 2018. Citado 2 vezes nas páginas 32 e 66.

27 BISONG, E. Google colaboratory. In: _____. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019. p. 59–64. ISBN 978-1-4842-4470-8. Disponível em: <https://doi.org/10.1007/978-1-4842-4470-8_7>. Citado na página 33.

28 LIU, D. C.; NOCEDAL, J. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, v. 45, n. 1, p. 503–528, Aug 1989. ISSN 1436-4646. Disponível em: <https://doi.org/10.1007/BF01589116>. Citado na página 37.

29 KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2014. Citado 2 vezes nas páginas 38 e 108.

30 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado na página 38.

31 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*. [S.l.]: JMLR.org, 2010. (JMLR Workshop and Conference Proceedings, v. 9), p. 249–256. Citado na página 38.

32 NOCEDAL, J. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, American Mathematical Society, v. 35, n. 151, p. 773–782, 1980. Citado na página 38.

33 LIU, D. C.; NOCEDAL, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, Springer, v. 45, n. 1-3, p. 503–528, 1989. Citado na página 38.

34 NOCEDAL, J.; WRIGHT, S. J. *Numerical Optimization*. Second. [S.l.]: Springer Science & Business Media, 2006. Citado na página 38.

35 MIRANDA, R. *MA311 - Cálculo III: Introdução às Equações Diferenciais Ordinárias*. [S.l.], 2024. Notas de aula. Disponível em: <https://www.ime.unicamp.br/~rmiranda/wp-content/uploads/2024/04/ma311-0-intro.pdf>. Citado 2 vezes nas páginas 38 e 39.

36 M., V. *Uma Breve Introdução as Equações Diferenciais: Alguns Tipos de Equações Diferenciais Ordinárias*. Dissertação (Mestrado) — Universidade Federal do Pará, 2014. Disponível em: <https://www.matematica.icen.ufpa.br/images/tccs/TCC_FACMAT_17.pdf>. Citado 2 vezes nas páginas 38 e 39.

37 B., N. *Equações Diferenciais Ordinárias de Primeira Ordem e Aplicações*. Dissertação (Mestrado) — Universidade Federal do Pará, Dezembro 2013. Disponível em: <https://matematica.icen.ufpa.br/images/tccs/TCC_FACMAT_06.pdf>. Citado na página 39.

38 BOYCE, W. E.; DIPRIMA, R. C. *Elementary Differential Equations 10E*. Chichester, England: John Wiley & Sons, 2012. Citado na página 40.

39 STRAUSS, W. A. *Partial Differential Equations: An Introduction*. 2nd. ed. Hoboken, NJ: John Wiley & Sons, 2007. Citado 2 vezes nas páginas 42 e 43.

40 HABERMAN, R. *Applied Partial Differential Equations with Fourier Series and Boundary Value Problems*. 4th. ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004. A 5th edition was published in 2013. Citado na página 42.

41 JOHN, F. *Partial Differential Equations*. 4th. ed. New York: Springer-Verlag, 1982. Citado na página 42.

42 AEROSTUDENTS.COM. *Partial Differential Equations Full Version*. Accessed May 25, 2024. Disponível em: <http://www.aerostudents.com/courses/partial-differential-equations/partialDifferentialEquationsFullVersion.pdf>. Citado na página 42.

43 EVANS, L. C. *Partial Differential Equations*. 2nd. ed. Providence, RI: American Mathematical Society, 2010. v. 19. (Graduate Studies in Mathematics, v. 19). Citado na página 43.

44 CVETIC, M.; GRIFFIES, S. *Partial Differential Equations. Chapter 6*. 1994. Lecture Notes, Georgia Institute of Technology. Accessed May 25, 2024. Disponível em: <https://cns.gatech.edu/~predrag/courses/PHYS-6124-12/StGoChap6.pdf>. Citado na página 43.

45 KLAINERMAN, S. *PDE as a Unified Subject*. Undated. Essay/Paper, Princeton University. Accessed May 25, 2024. Disponível em: <https://web.math.princeton.edu/~seri/papers/telaviv.pdf>. Citado na página 43.

46 FULLING, S. A. *Classification of Second Order Linear PDEs*. 2015. Lecture Notes, Texas A&M University. Accessed May 25, 2024. Disponível em: <https://people.tamu.edu/~fulling/m412/f15/classify.pdf>. Citado na página 43.

47 STROGATZ, S. H. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. 2nd. ed. [S.l.]: Westview Press, 2015. Citado na página 44.

48 TEMAM, R. *Navier-Stokes Equations: Theory and Numerical Analysis*. [S.l.]: AMS Chelsea Publishing, 2001. v. 2. (Studies in Mathematics and its Applications, v. 2). Citado na página 44.

49 OTT, E. *Chaos in Dynamical Systems*. 2nd. ed. [S.l.]: Cambridge University Press, 2002. Citado na página 44.

50 CAI, S. et al. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, v. 143, n. 6, p. 060801, 04 2021. ISSN 0022-1481. Disponível em: <https://doi.org/10.1115/1.4050542>. Citado na página 45.

51 RUSSELL, J. S. Report on waves. In: *Report of the fourteenth meeting of the British Association for the Advancement of Science*. [S.l.: s.n.], 1844. v. 25. Citado na página 45.

52 KORTEWEG, D. J.; VRIES, G. D. Xli. on the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor & Francis, v. 39, n. 240, p. 422–443, 1895. Citado na página 46.

53 RAJARAMAN, R. *Solitons and Instantons: An Introduction to Solitons and Instantons in Quantum Field Theory*. [S.l.]: North-Holland, 1989. Citado 2 vezes nas páginas 46 e 49.

54 Chapter 1 bäcklund transformations and their application to nonlinear equations of mathematical physics. In: ROGERS, C.; SHADWICK, W. (Ed.). *Bäcklund Transformations and Their Applications*. Elsevier, 1982, (Mathematics in Science and Engineering, v. 161). p. 12–122. Disponível em: <https://www.sciencedirect.com/science/article/pii/S007653920863074X>. Citado na página 50.

55 MAZIAR, R.; PERDIKARIS, P.; KARNIADAKIS, G. e. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *ArXiv*, v1, n. 1711.10561, p. 1–22, 2017. Citado 5 vezes nas páginas 54, 55, 68, 69 e 71.

56 MAZIAR, R.; PERDIKARIS, P.; KARNIADAKIS, G. e. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *ArXiv*, v1, n. 1711.10566, p. 1–19, 2017. Citado 2 vezes nas páginas 54 e 68.

57 NGUYEN, L. T. K. Modified homogeneous balance method: Applications and new solutions. *Chaos, Solitons & Fractals*, v. 73, p. 148–155, 2015. ISSN 0960-0779. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0960077915000247>. Citado na página 59.

58 BRATLEY, P.; FOX, B. L.; NIEDERREITER, H. Implementation and tests of low discrepancy sequences. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, ACM, v. 2, n. 3, p. 195–213, 1992. Citado na página 61.

59 CHO, W. et al. *Parameterized Physics-informed Neural Networks for Parameterized PDEs*. 2024. Disponível em: <https://arxiv.org/abs/2408.09446>. Citado 3 vezes nas páginas 71, 72 e 73.

60 KHANDELWAL, A. et al. $100k or 100 days: Trade-offs when pre-training with academic resources. 2024. Disponível em: <https://arxiv.org/abs/2410.23261>. Citado na página 120.