Bianca Almeida Corrêa

Solução numérica de equações diferenciais via redes neurais artificiais

Uberlândia, Minas Gerais, Brasil

Bianca Almeida Corrêa

Solução numérica de equações diferenciais via redes neurais artificiais

Trabalho de Conclusão de Curso em Física de materiais da **Universidade Federal de Uberlândia**, como requisito para obtenção do grau de **Bacharelado em Física de Materiais.**

Universidade Federal de Uberlândia – UFU

Instituto de Física – INFIS

Física de Materiais

Orientador: Prof. Dr. Gerson Ferreira Junior

Coorientador: Prof. Dr. Marco Aurélio Boselli

Uberlândia, Minas Gerais, Brasil

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

C824 Corrêa, Bianca Almeida, 1999-
2024 Solução Númerica de Equações Diferenciais via Redes Neurais Artificiais [recurso eletrônico] / Bianca Almeida Corrêa. - 2024.

Orientador: Gerson Ferreira.
Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Uberlândia, Graduação em Física de Materiais.
Modo de acesso: Internet.
Inclui bibliografia.

1. Física. I. Ferreira, Gerson,1982-, (Orient.). II. Universidade Federal de Uberlândia. Graduação em Física de Materiais. III. Título.

CDU: 53

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

Dedico este trabalho a todos os jovens físicos que se propuseram a está longa e difícil jornada que é a ciência, dedico aos meus pais, meu noivo e familiares e meus amigos que sempre me apoiaram neste desafio.

Agradecimentos

Gostaria de agradecer aos meus pais que nunca deixaram de me apoiar, a minha irmã, ao meu noivo, e a todos os meus familiares, aos meus físicamigos que embarcaram nesta jornada comigo e aos amigos que fui fazendo ao longo deste tempo. Agradecer também as agencias de fomento a pesquisa CAPES, CNPq, Fapemig e a ProPP.

"O importante é não é não parar de questionar; a curiosidade tem a sua própria razão de existir." – Albert Einstein

Resumo

Este trabalho visa aprender e explorar possíveis aplicações das redes neurais artificiais usando o método de redes neurais informadas pela física (PINN, do inglês, *Physics Informed Neural Network*). As PINNs são um método que utiliza o resíduo da equação diferencial a ser resolvida na função de erro e suas condições de contorno e condições iniciais. Neste contexto, as PINN's foram aplicadas no problema do oscilador harmônico a fim de verificar como a precisão do método varia com o número de neurônios artificiais, com diferentes funções de ativação e com diferentes números de épocas. Para isso, foi utilizado um código base que resolve o problema do oscilador harmônico amortecido usando a abordagem de PINN's. Após o estudo e entendimento das redes neurais, foram realizadas modificações de melhoria e testes de desempenho no código autoral para fins de estudo e prática. Durante a fase de testes, foi possível aplicar o método de aprendizado por sequência, onde foi obtido um bom resultado de convergência do modelo. E por fim, concluímos que, as redes neurais e as PINN's são uma abordagem inovadora e promissora, no entanto, dependendo do problema em que estão sendo aplicadas, os resultados gerados não demonstram uma convergência e eficiência que é esperada. Além disso, o tempo de execução das PINN's, quando comparado a métodos como Runge-Kutta, é alto. Enquanto a rede neural roda em minutos, runge-kutta roda em segundos, então vimos que a execução das PINN's comparada a outros métodos também não apresenta uma vantagem significativa.

Palavras-chave: aprendizado de máquina, redes neurais artificias e PINN's.

Abstract

This work aims to learn and explore possible applications of artificial neural networks using the Physics Informed Neural Network (PINN) method. PINNs, or physics-informed neural networks, are a method that uses the residual of the differential equation to be solved in the error function along with its boundary and initial conditions. In this context, PINNs were applied to the problem of the harmonic oscillator to assess how the method's accuracy varies with the number of artificial neurons, different activation functions, and different numbers of epochs. To achieve this, a base code solving the problem of damped harmonic oscillator using the PINN approach was utilized. After studying and understanding neural networks, modifications were made to improve and test the performance of the original code for study and practice purposes. During the testing phase, it was possible to apply the sequence learning method, which resulted in a good model convergence. Finally, we conclude that neural networks and PINNs represent an innovative and promising approach; however, depending on the problem they are applied to, the generated results may not demonstrate the expected convergence and efficiency.Furthermore, the execution time of PINNs, when compared to methods like Runge-Kutta, is high. While the neural network runs in minutes, Runge-Kutta runs in seconds, so we see that the execution of PINNs compared to other methods also does not present a significant advantage.

Keywords: machine learning, artificial neural network and PINN's

Sumário

1 Introdução

Atualmente, muito se investiga sobre a natureza da mente e como acontece o processo de tomada de decisão no cérebro humano aliada ao contexto de processos computacionais. Também conhecida como concepção computacional, ela identifica a mentalidade de execução do algoritmo de maneira que, idealmente, as máquinas devem operar semelhantemente ao cérebro humano.

Pioneiro nos estudos de inteligencia artificial e conhecido como pai da computação, o matemático britânico Alan Turing (1912-1954) foi responsável por propor, em destaque, o Teste de Turing (Turing Test), um método teórico criado para determinar se as habilidades das máquinas eram compatíveis com as habilidades dos seres humanos. O teste de Turing é um teste composto por três elementos, um avaliador, um individuo e um computador. O objetivo é que a pessoa avaliadora consiga descobrir com quem está falando, com o individuo ou com o computador. Ao passo que, a máquina foi treinada para responder como um humano e tem como objetivo confundir o avaliador. A conclusão do teste de Turing é, caso a máquina consiga induzir o avaliador ao erro, logo, a máquina é inteligente, e isso significa que ela é capaz de imitar o comportamento humano.

Publicado em 1950 o artigo Computing Machinery and Intelligence (1) instigava abordar sobre a capacidade das máquinas imitarem o pensamento humano por meio de testes comportamentais a fim de provar a existência, ou não existência da mentalidade da máquina. Eles concluíram que o principal problema é a parte da programação e que deverão haver avanços nas engenharias para mitigar este problema. Além disso, diz que as máquinas podem competir com os homens em todos os campos puramente intelectuais.

Atualmente, as evoluções tecnológicas e computacionais possibilitaram o desenvolvimento e o aprimoramento de ferramentas que automatizam, facilitam e reduzem o tempo gasto em processos industriais e científicos. Na busca por sistemas compatíveis a estes processos complexos, um exemplo em destaque, as redes neurais artificias tornaram-se uma opção válida. (2)

Frequentemente caracterizada como uma sub-especialidade da Inteligencia Artificial, as redes neurais artificiais, constituem uma área multidisciplinar para estudos e resolução de problemas complexos, tais como reconhecimento de padrões, classificação, otimização, estimativas e previsões, clusterização, dentre outras.

As redes neurais artificiais são sistemas desenvolvidos com intuito de imitar uma capacidade cognitiva em particular de um ser humano (3), em especial, quando acontece uma tomada de decisão.

Neste âmbito, uma rede neural artificial é uma máquina projetada para modelar a maneira como o cérebro realiza uma tarefa de interesse. Para serem eficientes, as redes neurais operam com interligações maciças de células computacionais, também conhecidas como neurônios ou unidades de processamento.(4)

Este trabalho visa apresentar uma breve introdução dos conceitos básicos de redes neurais artificias, sua arquitetura e suas principais características. Foi utilizado um trabalho como base para serem feitos testes e modificações com intuito de entender como a convergência do modelo é afetada pelas alterações nos hiperparametros da rede neural. Além disso, verificamos se realmente existe um ganho significativo na implementação das redes neurais em vista de outros métodos como runge-kutta.

1.1 Redes Neurais Artificiais

Inicialmente as redes neurais artificiais (RNA's, do inglês *Artificial Neural Network*) surgiram a partir de um modelo matemático baseado no neurônio biológico proposto em 1943 por Warren McCulloch e Walter Pitts (5). No artigo são feitos cálculos das redes neurais que possibilitaram unificar a parte biológica com a lógica matemática.

Para o desenvolvimento do artigo, eles se basearem em três pilares importantes: fisiologia básica e a atribuição dos neurônios no cérebro, análise formal da lógica criada por Russel e Whitehead e a teoria da computação de Turing (6).

A análise formal da lógica se baseia no fato de que, qualquer função que fosse computável, isso significa, funções que podem ser calculadas usando um dispositivo mecânico de cálculo dado uma quantidade ilimitada de espaço de armazenamento e de tempo de execução (7).

Essas funções computáveis poderiam ser calculadas por uma determinada quantidade de neurônios interconectados por conectivos lógicos, implementados com estruturas de rede simples.

Em 1949, Hebb explicitou a regra de atualização simples, também conhecido como aprendizagem de Hebb ou regra de Hebb (8), que tinha como objetivo modificar as intensidades das conexões entre os neurônios. (9). A regra de aprendizagem de Hebb tem como base o modelo de neurônio biológico, reescrita em duas partes:

1. Se dois neurônios em ambos os lados de uma sinapse (conexão) são ativados simultaneamente, então a força daquela sinapse é seletivamente aumentada.

2. Se dois neurônios em ambos os lados de uma sinapse são ativados assincronamente, então aquela sinapse é seletivamente enfraquecida ou eliminada.

As sinapses hebbianas são definidas como um mecanismo dependente do tempo, altamente local e fortemente interativo para aumentar a eficiência sináptica (4). A importância da regra de Hebb fica clara nos modelos de processo de aprendizagem das RNA's que serão discutidas mais a frente.

Até então, vários estudos sobre inteligência artificial já haviam sido desenvolvidos, porém, foi apenas em 1950 com o teste de Turing que as máquinas foram desenvolvidas baseadas em inteligência artificial. De fato, em 1951 foi criado o primeiro neurocomputador também conhecido como SNARC (do inglês, *Stochastic Neural Analog Reinforcement Calculator*) (10). Era um aparelho com capacidade de realizar operações matemáticas simulando sinapses.

Logo foi possível definir o neurônio artificial como sendo a unidade básica que constitui uma rede neural artificial composta por várias unidades de processamento além de três elementos essenciais:

1. Sinapses ou elos de conexão que são associadas a um peso sináptico w_{kj}, sendo o índice k o neurônio em questão e j se refere ao input que o peso sináptico está relacionado.

2. Combinador linear, responsável por somar os sinais de entrada, ponderados pelas respectivas sinapses do neurônio.

3. Função de ativação, resumidamente, é responsável por restringir a amplitude da saída de um neurônio.

Além desses elementos citados acima, ainda existem mais elementos que constituem a arquitetura de uma RNA's, tais como os inputs ou dados de entrada e bias. Eles podem ser escritos da forma vetorial como $\vec{x}$, o bias é uma variável que limita o grau de liberdade da função de ativação, é um parâmetro livre que pode ser reajustado de acordo com a necessidade. O sinal de entrada correspondente a sinapse i conectada a neurônio j, que é multiplicado por um peso sináptico escritos como uma matriz, da forma, w_{ij}. Então no neurônio, acontece a soma ponderada dos sinais de entrada com o bias, de forma que:

$$u_i^l = (b_i^l + \sum_j w_{ij}^l x_j^{l-1}) \tag{1.1}$$

e

$$y_i^l = \varphi(u_i^l) \tag{1.2}$$

Onde $\vec{y}$ é conhecido como sinal de saída e φ a função de ativação. No modelo que o artigo (5) cita, cada neurônio é alimentado pelos dados de entrada e caracterizado pelos

pesos, bias e função de ativação. Os neurônios realizam a transformação linear na entrada pelos pesos e bias e a informação se propaga da entrada para as camadas ocultas, as quais fazem o processamento da informação e a enviam para a camada final. Este processo também é conhecido como propagação direta (do inglês, *Feedfoward pass*).

1.1.1 Componentes das Redes Neurais

A construção de uma rede neural artificial é baseada na escolha ideal dos hiperparâmetros antes do treinamento. A rede é inicializada a partir dos dados de entrada, também conhecidos como inputs, e eles dependem diretamente da quantidade de dimensões que serão analisadas no problema. A mesma concepção pode ser aplicada na camada do output. Logo, para cada sistema a quantidade de inputs e de outputs deve ser definida antes do treinamento.

Em seguida, correspondente a importância relativa de cada input, os pesos sinápticos e o bias são ditos parâmetros livres que podem ser ou não serem definidos aleatoriamente. Os pesos sinápticos são como estímulos para ativação do neurônio, ou seja, quanto maior a sua importância relativa mais influencia terá na saída da camada. Já o bias, é uma variável que tem o objetivo de aumentar ou diminuir o grau de liberdade da função de ativação.

Então a rede vai receber os valores de entrada multiplicados pelos pesos sinápticos correspondentes e somados com o bias. Após o somatório, a informação é passada pela função de ativação, responsável por restringir a saída da informação dentro de um intervalo fechado.

As funções de ativação são hiperpârametros de configuração da rede que vão definir o valor de saída de cada neurônio. As mais comuns são do tipo: função sigmoide, função tangente hiperbólica e função ReLu (do inglês, *Rectified Linear Unit*).

A função de ativação transforma os dados de entrada de maneira não-linear devido às características não-lineares inerentes ao mapeamento das camadas da rede (2). Além disso,a função de ativação deve ser diferenciável para que seja possível o aprendizado de tarefas complexos. Acresce que, a função de ativação não-linear e diferenciável possibilita a retro-propagação devido ao cálculo do gradiente que reajustam os pesos e biais.

Após a função de ativação cumprir o seu papel, a saída da rede é quantificada pela função de perda. A função de perda refere-se a discrepância entre o valor predito e ao valor esperado, sendo assim, quanto menor o valor resultante da função melhor o resultado do modelo.

Dito isso, a função de perda é otimizada pelo método do gradiente descendente, que calcula as derivadas parciais da função de perda pelos parâmetros livres da rede (pesos e biais). Espera-se achar os valores mínimos possíveis que podem ser assumidos pela função. Após o cálculos, os parâmetros livres são reajustados para melhorar o desempenho do

aprendizado da rede e a convergência do resultado.

1.1.2 Funções de Ativação

A função de ativação funciona como modelo matemático disponível para controle de sistemas não-lineares, transformando a tarefa de controle não-linear em pequenas tarefas de controle linear. A função de ativação nas RNA's representa o efeito que os dados de entrada internos e o estado atual de ativação exercem na definição do próximo estado de forma que, estas funções permitem que mudanças nos pesos e bias resultem em alterações no resultado final. (4).

Das funções de ativação, os tipos básicos são: a função linear, a função sigmoide, a função tangente hiperbólica (tanh) e a função Relu (do inglês, *Rectified Linear Unit)*. Na figura 1, ilustra como é o mapeamento da saída das diferentes funções de ativação.

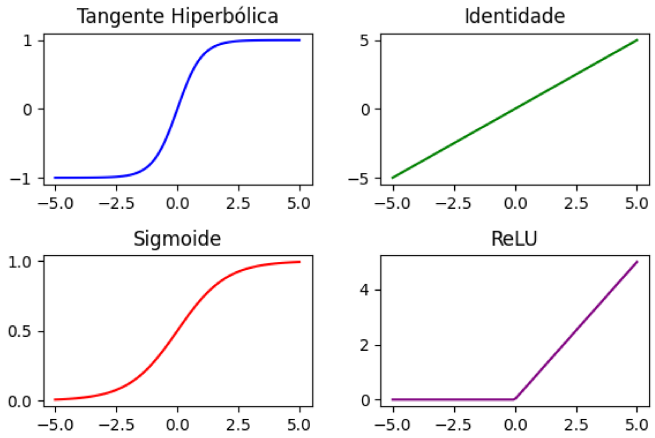

Figura 1 – Mapeamento das diferentes funções de ativação: função do tipo linear, a função do tipo sigmoide, a função do tipo tangente hiperbólica e a função do tipo Relu. Adaptado de (11)

A função tanh tem uma saída no intervalo (-1, 1) e pode saturar para valores extremos em situações em que as ativações são grandes. Isso pode levar a problemas onde o gradiente pode tender a valores muito pequenos ou a valores muito grandes, especialmente em redes neurais mais profundas (do inglês, *Deep Neural Network*), levando a um treinamento instável. As redes profundas, são redes que possuem várias camadas ocultas, que são um etapa intermediária entre a camada de entrada e a camada de saída, e que trabalham com uma grande quantidade de dados, além disso, possuem uma grande quantidade de neurônios interligados.

A função sigmoid mapeia entradas entre 0 e 1, o que pode reduzir a magnitude dos valores durante o treinamento, utilizada em classificação binaria. No entanto, ela é desvantajosa quando usada em redes com camadas muito profundas. Isto significa que a

propagação dos gradientes dos pesos e dos bias da rede durante o processo de treinamento podem tender a valores muito pequenos.

A função *ReLu* é mais comum em problemas de aprendizado profundo, pois, retorna o valor de entrada se o valor for positivo e zero se for negativo. É eficiente em termos computacionais e tem bom desempenho para tarefas de classificação e regressão.

1.1.2.1 Exemplo

Na Figura 2, é ilustrado como um mesmo modelo de rede neural simples pode apresentar mudanças significativas no resultado, em vista, da escolha da função de ativação. O modelo prediz o comportamento de uma parábola, uma função de segundo grau exata do tipo: $y = x^2$.

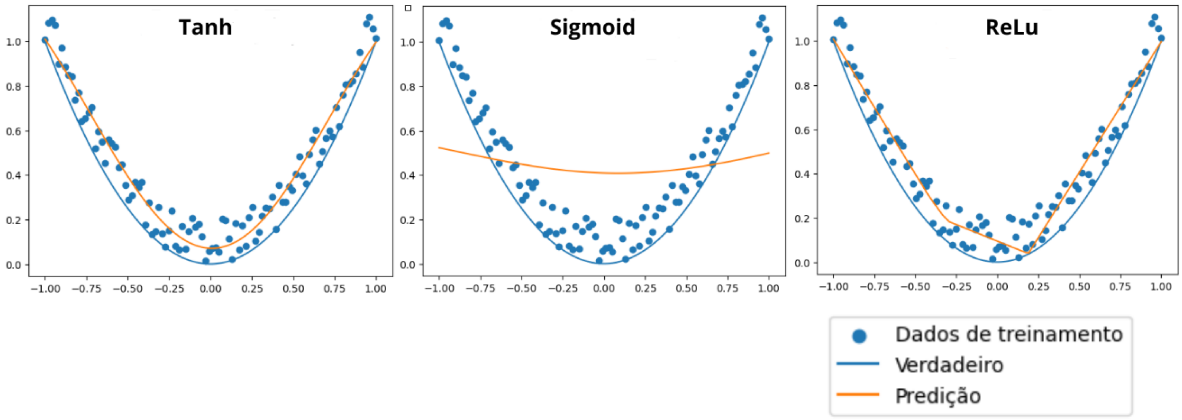

Figura 2 – Gráficos das diferentes funções de ativação: tanh, *ReLu* e *sigmoid*, respectivamente, usadas na mesma rede neural para reproduzir o comportamento de uma função parabólica.

A arquitetura da rede possui o mesmo conjunto de dados, o mesmo número de neurônios, o mesmo número de camadas, o mesmo número de épocas, a mesma função de perda, que mede a discrepância entre a saída do modelo e a solução esperada, o mesmo tamanho do passo e o mesmo otimizador para todos os gráficos da figura, a diferença de uma para outra é exclusivamente a função de ativação. De fato, é notável perceber que diferentes tipos de função de ativação podem afetar diretamente na convergência da rede neural, neste exemplo, a função tanh apresenta uma melhor concordância com os dados e com a solução exata que é representada pela curva em azul.

1.1.3 Tipos de Aprendizado

Após a definição da arquitetura da rede neural, acontece o processo de aprendizagem, um processo pelo qual os parâmetros livre de uma rede são adaptados por meio de estimulação pelo ambiente em questão (4). Os algoritmos de aprendizagem são baseados em regras bem definidas para a solução do problema de aprendizagem. Assim, o tipo de

aprendizagem é definido respeitando as modificações dos parâmetros e a necessidade do sistema em questão.

O processo de aprendizado é realizado por processos iterativos de ajustes aplicado aos pesos sinápticos e aos bias. Em resumo, o treinamento ajusta a matriz de pesos e bias, para que o vetor de saída coincida com o resultado esperado. E pode ser divido em aprendizado supervisionado e aprendizado não-supervisionado.

Durante o aprendizado supervisionado, um agente externo fornece a rede a saída desejada em relação ao padrão de entrada, de forma que, seja feita a comparação da predição com a resposta esperada. E a partir disso ajustam-se os parâmetros de pesos e bias com intuito de minimizar o erro a cada resposta gerada pela rede (2). Um exemplo de tarefa realizada por aprendizagem supervisionada é a regressão linear.

Já o aprendizado não-supervisionado, não existe um agente externo monitorando os resultado do treinamento, então a rede deve procurar por um padrão ou redundância nos dados e treinar a partir disso. Um exemplo é detecção de doenças via método de imagens médicas, o objetivo do sistema é descobrir os materiais diferentes que estão ilustrados na imagens e categorizá-los.

1.1.4 Forward Propagation

Após a escolha dos hiperparametros da rede acontece a propagação direta, (do inglês, *Forward propagation*) trata-se do processo de propagação dos dados de entrada através das camadas de entrada da rede até a camada de saída, gerando uma previsão ou classificação.

A passagem dos dados pelas camadas é um processo iterativo onde, as camadas são composta por neurônios, estes recebem os dados de entrada e aplicam a função de ativação. Depois de aplicada a função de ativação um valor de saída é gerado e propagado para próxima camada e esse processo de repete até a camada de saída.

Neste tipo de propagação de dados a informação flui em apenas uma direção, das camadas de entrada até a camada de saída, dessa forma, não existem conexões que retornem á camadas anteriores, ou seja, não há retroalimentação.

A propagação direta é especialmente utilizada em redes neurais de aprendizado profundo (do inglês, *Deep Learning*), que são caracterizadas por serem redes densas com várias camadas ocultas. Como a informação é processada de forma hierárquica, cada camada aprende a partir das informações aprendidas pelas camadas anteriores.

A propagação pode ser representada da seguinte forma:

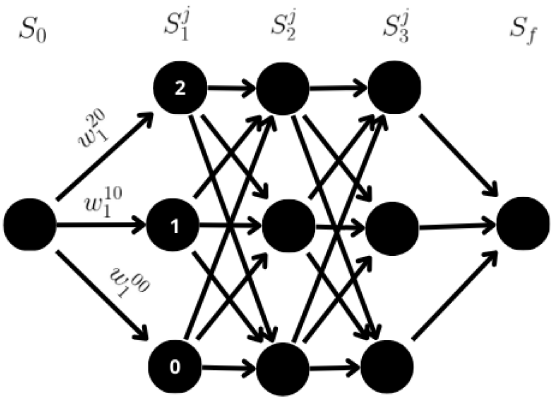

Figura 3 – Exemplo de rede neural com a passagem dos dados de entrada com propagação direta, no sentindo das setas considerandos os respectivos pesos sinápticos para cada dados de entrada.

$$\vec{S^j} = \begin{bmatrix} S_1^l \\ S_2^l \\ S_3^l \\ ... \\ S_i^l \end{bmatrix} \tag{1.3}$$

Onde S_i^l representa a saída de uma camada para outra, sendo l a camada (do inglês, *layer*) respectiva ao neurônio i. Os pesos (w), também podem ser escritos como matrizes, com l representando a camada e ij seu respectivo neurônio, onde i se refere ao input do neurônio anterior que vai j o próximo neurônio.

$$w_{ij}^l = \begin{bmatrix} w_{11}^1 & w_{12}^2 & ... & w_{1j}^l \\ w_{21}^1 & w_{22}^2 & ... & w_{2j}^l \\ ... & ... & ... & w_{ij}^l \end{bmatrix} \tag{1.4}$$

O parâmetro bias (b), também pode ser escrito como uma matriz coluna, com l representando a camada e i seu respectivo neurônio, onde i se refere ao input do neurônio anterior que vai j o próximo neurônio.

$$\vec{b_l} = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \\ ... \\ b_i^l \end{bmatrix} \tag{1.5}$$

Da mesma maneira, a função de ativação a pode ser escrito como uma matriz coluna, com l representando a a respectiva camada. Considerando uma função de ativação da forma $a(x)$, ela pode ser representada como:

$$a^l(x) = \begin{bmatrix} a^1(x) \\ a^2(x) \\ a^3(x) \\ \dots \\ a^l(x) \end{bmatrix} \tag{1.6}$$

Assim, associando o diagrama da figura 3, com a representação matricial, a propagação direta dos dados de entrada na i-ésima posição é da forma:

$$S_i^l = a^l(b_i^l + \sum_j w_{ij}^l S_j^{l-1}) \tag{1.7}$$

1.1.5 Backward Propagation

Diferente da propagação direta que não tem retroalimentação e é unidirecional, a retro-propagação ou propagação para trás (do inglês, *Backward propagation*) realimenta os valores calculados como entradas, novamente, o que acarreta a mudança no cálculo dos valores modificados na saída até a primeira camada, a figura 4 ilustra, pelas setas em azul, como acontece a retroalimentação das camadas.

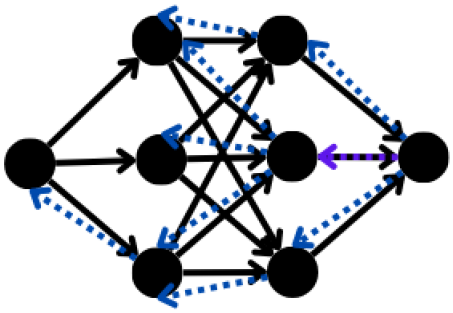

Figura 4 – Exemplo de rede neural com os sentidos da propagação direta representada pelas setinhas em preto (da esquerda para direita) e com o sentido da retro-propagação pelas setinhas pontilhadas em azul (da direita para esquerda). Adaptado de: (12)

No processo de propagação direta, como na figura 4 representado pelas setas em preto, os valores de pesos e bias são atribuídos aleatoriamente e propagados até a função de perda, que irá gerar o valor do erro associado ao processo de aprendizagem. É essencial que a rede tenha o menor valor relacionado ao erro. Dessa forma, quando o sinal de erro é gerado, o algoritmo tem a função de alterar os parâmetros do modelo para que a saída da rede resulte na saída desejada.

Seja $X^l = a^l(b^l + w^l x^{l-1})$ o output da rede, onde l representa as camadas, a^l representa a função de ativação, b^l o bias e w^l os pesos. A função de perda pode ser escrita como $C = (x^l - y)^2$, onde x^l é a saída da rede e y a solução desejada. Assim, para calcular o gradiente da função de perda, reescrevemos $X^l = a^l(z^l)$, onde $z^l = (b^l + w^l x^{l-1})$.

Como a propagação dos dados é atualizado a cada camada, o gradiente da função de perda deve levar em consideração os parâmetros livres, ou seja, os pesos e bias, assim como exemplificada na equação:

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial z^l_j}{\partial w^l_{jk}} \frac{\partial a^l_j}{\partial z^l_j} \frac{\partial C}{\partial a^l_j}. \tag{1.8}$$

Onde $j = l$, e $k = (l - 1)$ representam as camadas, porém trocando os índices para facilitar a visualização, a equação 1.8 representa a derivada da função de custo em relação ao peso da camada l em relação a jk. Da mesma maneira, a derivada para o ajuste do bias

Assim, os parâmetros livres da rede são reajustados de acordo com a correção do erro, e o sinal será propagado da saída para a entrada passando por todas as camadas.

1.1.6 Função de Perda e Gradiente

Afim de quantificar o resultado do treinamento da rede neural, a função de perda ou função de custo (do inglês, *Loss Function*) mede a discrepância entre a previsão da rede e o resultado esperado. O fato é, quanto menor o valor da função melhor é o resultado da predição.

A função de perda é escolhida de acordo com o problema que está sendo tratado, para problemas de regressão é comum a utilização do erro quadrado médio (do inglês, *Mean Squared Error*):

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - Y_i)^2 \tag{1.9}$$

E para problemas de classificação a utilização da entropia cruzada (do inglês, *Cross-Entropy*):

$$L = -\frac{1}{n} [\sum_{j=1}^{N} [t_j log(p_j) + (1 - t_j) log(1 - p_j)]] \tag{1.10}$$

Neste caso, quando a predição retorna valores próximos de 0, significa que a convergência do modelo está sendo feita corretamente, caso seja mais próximo de 1 significa que o modelo está errando.

Durante o treinamento, o objetivo é minimizar a função de perda ajustando os pesos e bias de acordo com o algoritmo de otimização. Geralmente, os otimizadores encontram o

gradiente descendente da função, afim de que, o valor mínimo da função de perda seja encontrado mais rapidamente.

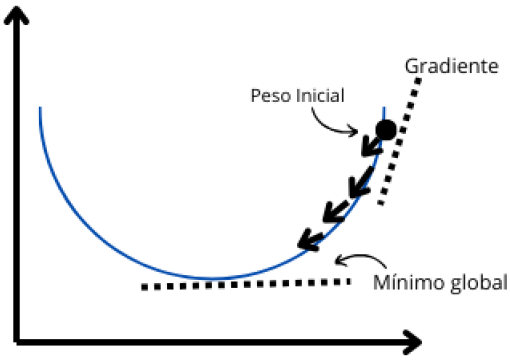

Figura 5 – Exemplo gráfico do sentido do gradiente descendente buscando o mínimo global da função em estudo. Adaptado de: (13)

O gradiente descendente calcula a derivada parcial da função de perda em relação aos pesos e bias da rede e reajusta buscando valores de pesos e bias que minimizem a função de perda.

O cálculo para minimizar a função perda ε consiste na repetição da regra da cadeia, as derivadas parciais são feitas para todos parâmetros livres da rede, os pesos sinápticos e os bias.

A taxa de aprendizado ou tamanho do passo, é outro parâmetro da rede, é um escalar que multiplica o gradiente para determinar o próximo passo de iteração. Se o valor da taxa de aprendizado for pequena, o treinamento será feito de forma mais sútil e sem tantos saltos grandes, porém o tempo de execução será maior. Caso o valor da taxa seja grande, o otimizador pode ultrapassar o mínimo que estava tentando encontrar e assim o treinamento não irá convergir adequadamente.

Ainda sobre o gradiente descente, existem diferentes tipo de algoritmos de otimização para o gradiente descendente, dentre eles o Adam (do inglês, *Adaptive Moment Estimation*). O algoritmo Adam combina técnicas em um algoritmo de aprendizagem eficiente, é aplicável quando a função é diferenciável em relação aos parâmetros livres da rede. (14)

Isso é a chamada aprendizagem por retro-propagação (do inglês, *backward propagation*). Ela é descrita em dois passos: o primeiro passo é a propagação para frente, neste passo os pesos e bias são todos fixos. No segundo passo, durante a retro-propagação, os pesos e bias são ajustados de acordo com a correção do erro. (15)

Isso significa que, a predição da rede é subtraída da resposta desejada para produzir

um sinal de erro, e esse sinal de erro é então propagado para trás através da rede, contra a direção das conexões sinápticas.(16) A fim de reajusta-los para minimizar a função de perda produzida pela rede.

1.1.6.1 Autograd

O autograd é um pacote que é responsável pelo cálculo da diferenciação automática em códigos pyhton e numpy, inclusive, em códigos que contém loops, condições, classes e até mesmo calculo de gradiente (17). É também utilizado como pacote da biblioteca PyTorch usado para calcular os gradientes dentro da rede neural.

O poder computacional do autograd deve-se ao fato de que ele é capaz de resolver o calculo do gradiente dinamicamente. Isso significa que, caso o modelo tenha ramificações de tomadas de decisão ou loops, o calculo ainda será feito corretamente e os resultados dos gradientes ainda estarão preciso.(18)

Durante a propagação direta, o autograd registra todas as operações em um tensor para gradiente e cria um grafo para encontrar a relação entre o tensor e todas as operações.(19) Esse conjunto de operações é chamado de diferenciação automática ou direta, além disso, este mecanismo é capaz de potencializar o treinamento da rede neural. (20) e (21)

O autograd é capaz de implementar o algoritmo de retro-propagação de forma eficiente, calculando os gradientes, numericamente, da função de perda. Ou seja, durante a etapa de retro-propagação, o autograd usa a regra da cadeia para calcular os gradientes da função de perda em relação aos pesos e bias.

A retro-propagação é inciada com o cálculo dos gradientes da função de perda em relação aos pesos e bias. Em seguida, esses gradientes são propagados de volta camada por camada, então, o autograd usa a regra da cadeia para calcular os gradientes em relação aos parâmetros de cada camada. Isso significa que, os gradientes da função de perda em relação a saída da camada são multiplicados pelos gradientes da saída da camada em relação aos parâmetros da camada. Ao final do cálculo do gradiente em relação a todos os parâmetros e camadas, os pesos e bias são ajustados.

Relembrando que, ao treinar o modelo o objetivo é minimizar a função de perda, logo a otimização da função é realizada reajustando os pesos e bias, de maneira que, a perda seja zero para todas as entradas (21). Então, as derivadas parciais da saída do modelo em relação as entradas envolvem muitas derivadas. Isto é, a derivada parcial será a soma dos produtos do gradiente local de cada caminho possível para encontrar o mínimo.

Dessa forma, a quantidade de derivadas correspondentes a cada caminho tenderá a valores grandes, especialmente em redes profundas. Com isso, o autograd é capaz de encontrar o histórico de cada cálculo, ou seja, cada tensor criado no modelo tem um

histórico de informações sobre os tensores de entrada e as funções usadas na sua criação (21).

1.2 Tipos de RNA's

A primeira RNA's foi proposta por Rossenblatt em 1958 (22), ele desenvolveu uma rede de múltiplos neurônios do tipo discriminadores lineares também conhecido como perceptron.

O perceptron é a forma mais simples de uma RNA's feedforward que dispõem de apenas um neurônio, cuja principal aplicação se da nos prolemas de classificação de padrões.(23) As redes do tipo perceptron de camada única são uteis para solução de problemas que admitem padrões linearmente separáveis.

Sempre que o perceptron for treinado com padrões que são de duas classes ditas linearmente separáveis, a rede será capaz de convergir para uma superfície de decisão formada por essas duas classes.(22)

Além do perceptron de camada única, existe também o perceptron multicamadas (MLP, do inglês, *Multilayer Perceptron*), ele é semelhante ao perceptron de camada única devido as suas características de detecção de padrões e classificações (24), mas sua arquitetura possui mais camadas ocultas.

Atualmente os perceptron multicamadas constituem três tipos de camadas de neurônios, sendo elas: uma camada de entrada, uma camada de saída e camadas ocultas. As redes perceptron multicamadas utilizam o algoritmo de retro-propagação.

Este algoritmo basicamente consiste na generalização do algoritmo de treinamento do perceptron de uma única camada por meio da estimação de erros e ajuste de parâmetros. O algoritmo é resolvido em três etapas: a propagação do sinal de entrada, a retro-propagação do erro da saída obtida em relação a saída desejada e o ajuste dos pesos e bias associados as conexões.

Além dos perceptrons, existem também as redes neurais recorrentes (RNN, do inglês, *Recorrent Neural Network*). É um tipo de RNA's que usa dados sequenciais ou dados de serie temporais, são comumente utilizados para reconhecer padrões de sequência de dados tais como textos, tradução de idiomas, reconhecimento de fala e legendagem de imagens. (25)

Diferente dos tipos de rede que assumem que os dados de entrada e saída são independentes uns dos outros, as redes recorrentes dependem dos elementos anteriores dentro da sequência, ou seja, há um dependência temporal entre os dados de entrada.(26)

As redes recorrentes possuem também pelo menos um laço de realimentação, isso significa que após aplicar uma nova entrada, a saída é calculada e então realimentada para

modificar a próxima entrada e assim sucessivamente.(6)

Outro tipo de rede neural, são as auto-encoder (AE), a arquitetura desta rede tem que ser adaptada para representar os dados de entrada em um espaço dimensional menor. (27) Isto significa que, a camada escondida deve possuir uma quantidade de neurônios menor do que a camada de entrada e a camada de saída é uma copia da camada de entrada.

As redes auto-encoder são dividas em duas partes: o codificador(encoder) e o decodificador (decoder). O codificador é responsável por compactar as entradas em um espaço latente, ou seja, em um espaço codificado, e o decodificador reconstrói a entrada da representação do espaço latente.

O processo de aprendizagem da rede auto-encoder é automatizado a partir de uma quantidade de dados, isso significa que é fácil treinar o algoritmo. Além disso uma aplicação comum do uso dessas redes é o reconhecimento de imagens e diminuição dos ruídos dos dados. Devido a estas características, as auto-encoder tornam-se interessantes para serem utilizada em imagens medicas computadorizadas, como por exemplo em exames de tomografia. (28)

Um tipo de rede neural que está sempre presente no dia-a-dia com recomendações de filmes, musicas e produtos e na trajetória mais eficiente. A máquina de Boltzmann (do inglês, *Boltzmann machine*), tem a arquitetura da rede neural tradicional, camadas de entrada, camadas ocultas e camadas de saída, e é um tipo de rede neural recorrente estocástica. (29)

Sua peculiaridade é o algoritmo de aprendizado não-supervisionado por contraste, que é responsável por reconhecer os padrões nos dados de entrada, mesmo com um grande volume e muitas variáveis nos dados. Como suas aplicações são voltadas para otimização, reconhecimento de padrões e recomendação, algumas áreas, em especial a financeira incorporaram á maquina de Boltzmann para detectar tendencias e padrões nos preços do mercado de ações. (30)

Diferente dos exemplos de redes neurais citados anteriormente, as redes neurais convolucionais (CNN, do inglês, *Convolutional Neural Network*) possuem uma arquitetura bem diferente. Nas camadas de convolução, as informações geradas anteriormente passam por um tipo de filtragem que acentuam os padrões regulares locais e assim a dimensão dos dados de entrada vão diminuindo a medida que a rede vai aprendendo.

Por essas características de detecção de padrões e classificação, as redes convolucionais são comumente utilizadas em classificação de imagens, reconhecimento de objetos e visão computacional. Um exemplo de aplicação desta rede, em especial, é na classificação de tuberculose pulmonar, propiciando grande precisão nos diagnósticos. (31) (32)

E por fim, o tipo de rede que será explorada neste projeto são as redes neurais

totalmente conectadas (do inglês, *Fully Connected*). Em particular, a vantagem neste tipo de rede está relacionada ao fato de que, arquiteturas de redes totalmente conectadas são ditas como aproximadores universais.(33)

Assim, é notável como os diversos tipos de RNA's estão presentes no cotidiano, e como suas aplicações abrangem múltiplas áreas e facilitam a resolução de problemas complexos.

1.3 Redes Neurais Informadas pela Física (PINN'S)

Entre as principais características das RNA's destacam-se a capacidade de modelagem de problemas complexos, a detecção e classificação de padrões, e a otimização, o que tem viabilizado sua implementação em diversas áreas do conhecimento. Em destaque, a otimização e o avanço no desempenho em problemas relacionados a física.

No entanto, o custo e o tempo da aquisição dos dados para aprendizagem do modelo muitas vezes é desafiador.(34) E trabalhar com uma quantidade insuficiente de dados para modelar problemas científicos complexos, na maioria das vezes, gera resultados generalizados e consequentemente não garantem a convergência do modelo.

Graças ao problema de generalização, emergiu um campo revolucionário conhecido como aprendizado de máquina cientifico (do inglês, *Scientific Machine Learning - SciML*). O objetivo principal deste aprendizado de maquina é combinar os conhecimentos científicos com o aprendizado de máquina, gerando algoritmos poderosos que aprendem pelo conhecimento prévio. (35)

Neste contexto, algumas tarefas em aprendizado de máquina (do inglês, *Machine Learning*) associam o modelo computacional a sistemas que tenham equações governantes ou fórmulas baseadas em leis da física replicando o sistema ao modelo.

As leis físicas, em sua maioria, podem ser descritas por modelos matemáticos conhecidos, principalmente por equações diferenciais (ED). As equações diferenciais são definidas como equações em que a incógnita é uma função de uma ou mais variáveis, envolvendo derivadas da variável dependente em relação a uma ou mais variáveis independentes.(36)

No entanto, para resolver problemas complexos as soluções das equações diferenciais podem não ser triviais, neste caso, a abordagem computacional funciona como uma alternativa poderosa e promissora. (37)

A junção entre equações governantes e as técnicas científicas de aprendizado de máquina, formaram as PINN'S (do inglês, *Physics Informed Neural Network*). De forma sucinta, as redes neurais informadas pela física adicionam novos termos na função de perda, termos que estão relacionados ao resíduo da equação diferencial, suas condições de contorno e condições iniciais (38).

Isso significa que as PINN's aproximam as soluções das equações diferenciais. A rede neural é treinada para minimizar a função de perda, incluindo termos que se referem as condições iniciais e condições de contorno e o resíduo das equações diferenciais dentro do domínio.(39)

As PINN's tiveram maior visibilidade a partir de 2017, com a publicação dos artigos (40) e (34). O artigo (34) é divido em duas etapas: para solução orientada a dados e descoberta de equações diferenciais parciais orientadas a dados.

Ao final, foi mostrado que essas redes neurais informadas pela física podem ser usadas para inferir soluções para as equações diferenciais obtendo modelos que são diferenciáveis para todos as coordenadas de entrada e parâmetros livres.E também ele apresenta uma nova classe de funções que são chamadas de aproximadores universais(34) .

Em uma PINN's, o tipo de aprendizado é chamado semi-supervisionado visto que, não é necessário que a solução analítica seja conhecida. Neste tipo de modelo, utiliza-se a diferenciação automática para calcular as derivadas parciais em relação aos parâmetros livres da saída da rede.

A partir do cálculo dessas derivadas parciais, é possível descrever a equação alvo para ser utilizada para o cálculo da função de perda.(41) Assim, para satisfazer a condição imposta, o algoritmo deve ser capaz de modelar o sistema ajustando os dados enquanto reduz o resíduo da equação diferencial.

Dito isso, é importante ressaltar trabalhos que foram desenvolvidos usando PINN's, tais como resolver problemas clássicos conhecidos de física. Trabalhos que estudam a resolução de problemas de física quântica, como a equação de Schrödinger, de física clássica como a equação de onda e problemas de mecânica de fluídos.

Além dos trabalhos exploratórios que revisam a literatura sobre as PINN's, caracterizam suas vantagens e desvantagens e incluem diferentes tipos de redes neurais informadas pela física.

Na referência (39), é fornecido uma revisão da literatura sobre as PINN's, o estudo neste artigo também busca caracterizar as suas vantagens e desvantagens. Além disso explora trabalhos que envolvem diferentes tipos de PINN's, tais como, redes neurais restritas pela física (do inglês, *Physics-Constrained Neural Network*, PCNN), redes neurais conservativas (do inglês, *Conservative PINN*, CPINN).

O autor, além de mostrar os diferentes tipos de PINN's, mostra também que os estudos recentes tem dado ênfase em modificar a arquitetura da rede. Testando diferentes tipos de funções de ativação, diferentes técnicas de otimização do gradiente, diferentes estruturas aplicadas a função de perda.

Em (42), os autores propuseram usar a PINN's para resolver um problema clássico

da física, a equação de onda. A escolha do problema apresenta desafios devido a natureza propagativa e oscilatória que compõem a solução da equação de onda. Neste trabalho, a equação diferencial da onda e uma condição de contorno com restrições são usadas para que a rede neural aprenda a solução.

Ao final do trabalho, conseguiram concluir que a rede neural informada pela física é capaz de resolver o problema proposto fora de seus dados de treinamento limite. Isto é, a rede consegue minimizar os problemas de generalização que provem do aprendizado profundo (do inglês, *Deep Learning*). Perceberam que a rede foi capaz de generalizar sobre as condições iniciais, e assim, eliminando a necessidade de re-treinamento para cada solução. Em vista das simulações numéricas tradicionais, como método das diferenças finitas, a rede apresentou melhores resultados.

Além da equação de onda, outro problema clássico da física quântica é resolver a equação: $H|\psi(r,t)\rangle = i\hbar\frac{\partial}{\partial t}|\psi(r,t)\rangle$, conhecida como a equação de Schrödinger. Em (43), é proposto resolver a equação de Schrödinger não linear no problema do Condensado de Bose-Einstein. O resultado do aprendizado foi efetivo, pois, foram obtidos bons resultados que superaram outros métodos em termos de complexidade e desempenho computacional.

Como as PINN's têm sido eficientes na resolução de problemas complexos, tal que, viabilizou sua utilização em vários outros contextos, em especial na mecânica de fluidos como em (44). Neste trabalho, as PINN's podem ser uma alternativa mais viável para substituir experimentos caros e simulações demoradas em grande escala (45).

1.4 Oscilador Harmônico

Um problema clássico da física de oscilações que possui solução analítica bem definida, e será explorado neste trabalho, é o oscilador harmônico. O oscilador harmônico pode ser descrito como um sistema que, ao ser deslocado da sua posição de equilíbrio, sofre uma força restauradora proporcional ao seu deslocamento.

O oscilador harmônico pode ser caracterizado de três tipos de sistemas diferentes. Sendo eles: movimento amortecido, movimento sub-amortecido e movimento super amortecido. (46) A diferença de cada tipo de oscilação depende do coeficiente de amortecimento relacionado os sistema em questão.

Pela segunda Lei de Newton, o equilíbrio das forças em um sistema de oscilador harmônico simples e amortecido é descrita pela equação diferencial da forma:

$$\frac{d^2x}{dt^2} + 2cw_0^2\frac{dx}{dt} + w_0^2x = 0 \tag{1.11}$$

Onde w_0 representa a frequência natural de oscilação do sistema, a frequência de oscilação $w = w_0\sqrt{1-c^2}$ e c é chamado de coeficiente de amortecimento. Neste sistema, o

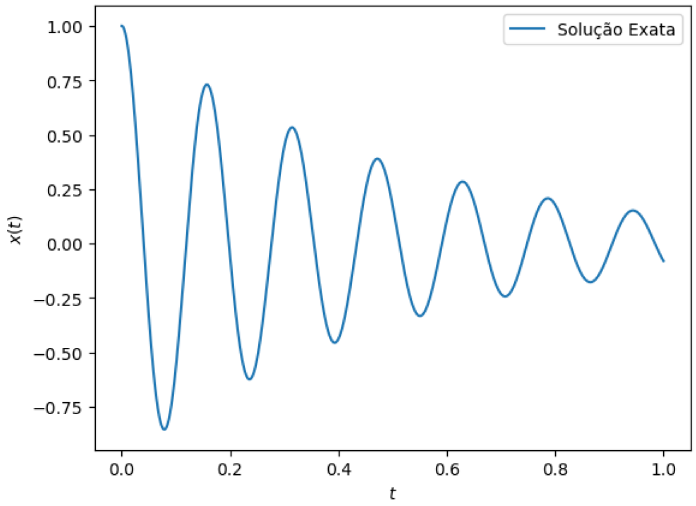

Figura 6 – Gráfico da evolução temporal da posição da solução exata do oscilador harmônico amortecido

coeficiente de amortecimento para o caso amortecido é tal que $c < 1$. E as condições de contorno para posição inicial e velocidade inicial, respectivamente:

$$x(0) = x_0 = 1 \tag{1.12}$$

$$v(0) = v_0 = 0 \tag{1.13}$$

Na figura 6, o gráfico representa a evolução temporal do deslocamento do oscilador harmônico amortecido, foram usados parâmetros de frequência natural $w_0 = 40$, a constante de amortecimento igual a $c = 0.05$, as condições iniciais de posição $x_0 = 1$, velocidade $v_0 = 0$ em um tempo $t_{max} = 1$.

Neste trabalho será considerado o caso do movimento amortecido. Assim, o oscilador harmônico amortecido corresponde a um sistema que acontece a dissipação de energia, seja devido a resistência do ar, atrito ou viscosidade de um líquido. Então a energia dos sistema é quantificada considerando uma força externa, oposta ao sentido do movimento, que fará com que a frequência de oscilação diminua ao passar do tempo.

Como dito anteriormente, as PINN's podem ser usadas como ferramenta para resolver equações diferenciais.

Na representação da rede neural na figura 7, os dados de entrada no primeiro neurônio é uma lista de valores de tempos (t). Esses dados são propagados através da rede seguindo a orientação das setas, levando em consideração os respectivos pesos, bias e funções de ativação de cada camada e de cada neurônio.

Durante este processo é feito o cálculo de x(t), que também é uma lista correspondente a posição a cada instante de tempo. O cálculo é realizado dentro da rede, cuja

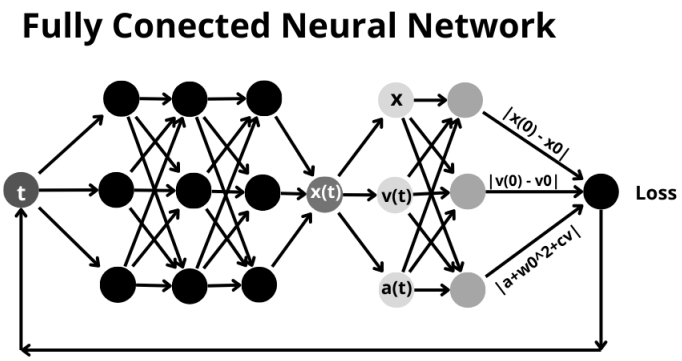

Figura 7 – Exemplo de Rede Neural totalmente conectada utilizada para solução do oscilador harmônico amortecido, aqui o input são os tempos que ao passar pela rede, vão gerar um $x(t)$, e o autograd vai ser usado para calcular as derivadas exatas de $x(t)$ para compor a função de perda com elementos como o resíduo da equação diferencial e as condições iniciais.

topologia, neste exemplo da figura 7, é representada por 3 camadas contendo 3 neurônios em cada.

Como o objetivo é minimizar o erro da predição, acontece o reajuste dos pesos e bias via retro-propagação. Como método do autograd, que será discutido a frente, utiliza as derivadas que são feitas durante a retro-propagação, correspondente a velocidade $v = \dot{x}(t)$, a aceleração $a = \ddot{x}(t)$ e a posição x(t).

Significa que são as derivadas do output (x(t)) pelo input (t), em seguida, nas penúltima camada, as derivadas são utilizadas para calcular os resíduos da equação diferencial que são usados na função de perda da rede neural, da forma:

$$L_1 = |x(0) - x_0|^2 \tag{1.14}$$

$$L_2 = |v(0) - v_0|^2 \tag{1.15}$$

$$L_3 = |\frac{\partial^2 x(t)}{\partial t^2} + w_0^2 x + c\frac{\partial x(t)}{\partial t}|^2 \tag{1.16}$$

E por fim, a perda (em inglês, *Loss*) é calculada pela média ponderada entre as perdas individuais: $Loss_{total} = c_1 L_1 + c_2 L_2 + c_3 L_3$ e c_n são constantes que balanceiam a importância relativa que está correlacionada a cada perda individual.

Em suma, a o treinamento da rede neural totalmente conectada utiliza os métodos de propagação direta, retro-propagação e minimização do erro. Além disso também será explorada a biblioteca de aprendizado de máquina de código aberto *Pytorch*.

2 Metodologia

2.1 Implementação

Para a implementação do modelo criado, foi utilizada a linguagem python e a biblioteca de aprendizado de máquina Pytorch. A biblioteca pytorch é acessível em termos de linguagem computacional, isso significa que, não é necessário aprender uma nova linguagem de programação. É especializada em diferenciação automática e cálculo de tensores, os tensores pytorch são matrizes multidimensionais que são usadas para todos os cálculos.

Dada a motivação referente as redes neurais artificiais, o método das PINN's e o problema do oscilador harmônico, foi implementado um modelo de rede neural informada pela física, baseada no trabalho desenvolvidos por Ben Mosely,(47). O autor utiliza o problema massa-mola para exemplificar o uso da PINN's, e com isso, explicar como o resíduo da equação diferencial é utilizada na função de perda da rede neural.

Dessa forma, este trabalho utiliza como base o que foi desenvolvido em (47), o código original foi adaptado a fim de testar melhorias, de analisar o comportamento da rede neural e principalmente, de aprender o funcionamento da rede neural informada pela física.

2.2 PINN

A rede neural informada pela física que foi desenvolvida neste trabalho está representada pela Figura 8. Os dados de entrada correspondem a uma lista de tempos, que são propagados pela rede considerando os pesos e bias aleatórios, e a função de ativação do tipo tangente hiperbólica.

Durante este processo de propagação direta a rede faz o cálculo de $x(t)$, que também é uma lista correspondente as posições em cada instante de tempo. A propagação dos dados acontece dentro de uma rede cuja topologia está exemplificada na figura 8, é o número de neurônios igual a 5 e o número de camadas igual a 2.

Além do processo de propagação direta, também acontece o processo de retro-propagação que visa minimizar o erro da predição, em seguida, reajustando os parâmetros livres da rede.

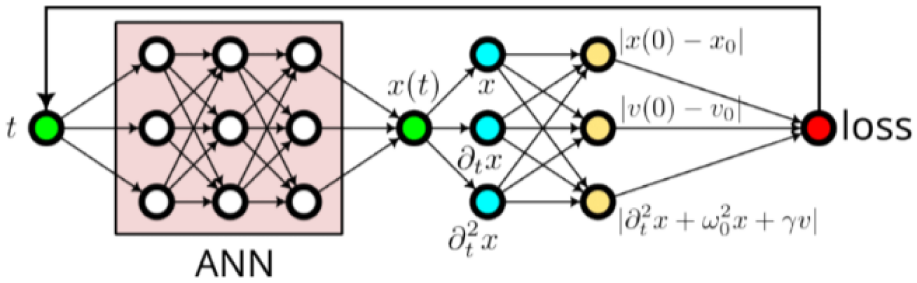

Figura 8 – Topologia da rede neural informada pela física que foi utilizada neste trabalho, a rede é inicializada com uma lista de tempos, essa lista de tempo é passada para a ANN onde são feitos cálculos levando em consideração os pesos e bias aleatórios, inicialmente, e a função de ativação, após isso, é gerado uma saída $x(t)$, o autograd então vai calcular as derivadas para a velocidade e aceleração, que vão compor a função de perda que é a soma da contribuição do resíduo da equação diferencial mais as condições iniciais

Neste contexto de PINN's, os cálculos da função de perda possuem uma sutileza, além de considerar as condições inciais do sistema, também leva em consideração o resíduo da equação diferencial.

$$L_1 = |(\ddot{x} + 2cw_0\dot{x} + w_0^2 x)|^2 \tag{2.1}$$

$$L_2 = |x(0) - x_0|^2 \tag{2.2}$$

$$L_3 = |v(0) - v_0|^2 \tag{2.3}$$

Onde, L_1 e L_2 são as perdas relacionadas as condições iniciais da posição e da velocidade, que são calculadas pelo erro quadrático médio. E L_3 está relacionada a equação diferencial do oscilador harmônico amortecido, onde $\ddot{x}$ é a aceleração, $\dot{x}$ é a velocidade e x a posição que são calculados pelo método do autograd de diferenciação exata.

2.3 Arquitetura da rede

A topologia da rede é ditada pelos hiperparâmetros, que são escolhidos antes do treinamento, dentre eles, o número de camadas, o número de neurônios, o número de épocas, a taxa de aprendizado, o otimizador e o tamanho dos lotes.

Na configuração de arquitetura da rede são escolhidos o número de neurônios, o número de camadas e a função de ativação. Neste trabalho, foi fixado, inicialmente, o número de neurônios (do inglês, *Neurons*) igual a 5, o número de camadas (do inglês, *Layers*) igual a 2, o tamanho da camada de entrada e a camada do output igual a 1 e a função de ativação do tipo tangente hiperbólica (tanh).

Na configuração de treino da rede é definido o tamanho do lote (do inglês, *Batch Size*), o número de épocas, a função de perda e o otimizador, e a taxa de aprendizado.

Neste trabalho, o tamanho do lote é igual a 100, o número de épocas é igual a 20000, a função de perda total é a soma do erro quadrático médio, para as condições inciais x_0 e v_0, mais a o erro da equação diferencial, o otimizador utilizado é o Adam e a taxa de aprendizado é da ordem de $lr = e^{-3}$.

Entre os hiperparametros citados, o tamanho da rede neural e o número de pontos de treinamento se relacionam de maneira direta e são essenciais para compreensão do desempenho e eficácia do aprendizado. É esperado que, quanto maior o tamanho da rede, maior a capacidade do modelo aprender os padrões nos dados de treinamento, sendo assim, a quantidade de dados deve ser proporcional.

No entanto, uma quantidade excessiva de dados de treinamento podem ocasionar o chamado *overfiting*, e uma quantidade insuficiente pode ocasionar o *underfitting*. Além disso, a quantidade de parâmetros da rede influência na relação entre o tamanho da rede e o número de pontos de treinamento. Quanto mais parâmetros o modelo tem, mais dados são necessário para ajusta-los corretamente.

O número de parâmetros pode ser calculado a partir dos elementos da rede neural, neste trabalho, os parâmetros da rede são os pesos sinápticos ($\vec{w^l}$) e bias ($\vec{b^l}$). Os dados propagados pela rede podem ser escritos na forma vetorial como:

$$\vec{S^l} = a^l(\vec{b^l} + w^l \vec{S^{l-1}}) \tag{2.4}$$

Considerando N o número de neurônios por camada, C o número de camadas e P o número de parâmetros é tal que:

$$\begin{aligned} P &= N + N^2(C-1) + N + 1 + NC \\ &= 1 + (C+2)N + N^2(C-1) \end{aligned} \tag{2.5}$$

Então o número de dados de treinamento ($batch$) deve ser expressivamente maior do que o número de parâmetros da rede, como: $batch = 10P$, para que aconteça a convergência adequada da rede neural.

Acresce que, com o aumento dos pontos de treinamento, a curva de aprendizagem da rede tende a se estabilizar, isso significa que, aumentar o número de pontos de treinamento pode ter bons resultados. Mas, possivelmente, está melhoria pode diminuir ao passo que a rede aprende a generalizar os dados.

2.4 Funções de ativação

Um dos principais elementos das redes neurais são as funções de ativação, resumidamente, elas são responsáveis por restringir o valor de saída de cada neurônio para a

próxima camada. É importante ressaltar que a não-linearidade da função de ativação deve ser suave, ou seja, diferenciável.

No contexto de redes neurais, as funções de ativação que são mais comuns são: a função sigmoide ou logística, a função tangente hiperbólica e função linear retificada (ReLu).

A função sigmoide é biologicamente semelhante a ativação do neurônio biológico, isto é, tem uma saída binária (ativa ou não ativa). A função sigmoide mapeia valores apenas entre 0 (não ativação) e 1 (ativação), além disso, a propagação do gradiente descendente, durante o processo de aprendizagem pode encontrar dificuldades, pois, as derivadas vão diminuindo e o gradiente tende a valores muitos pequenos.

A função ReLu (do inglês, *Rectified Linear Unit)* são fáceis de serem otimizadas, retorna zero em metade do seu domínio, isso gera consequências na tomada do gradiente descendente pois as derivadas se mantém altas enquanto está ativada.

Neste trabalho foi usado a função de ativação do tipo tangente hiperbólica (tanh) em todas as camadas da rede construída. A função tangente hiperbólica é similar a função sigmoide, outro tipo de função comumente usada como função de ativação, mas difere quanto a limitação de saída . A função tangente hiperbólica, preserva a forma sigmoidal da função de ativação sigmoide ou logística, mas mapeia a saída em valores de -1 a 1.

3 Resultados

Após o estudo da literatura de redes neurais e PINN's, seguimos para a parte de aplicação, neste trabalho, a aplicação é realizada no problema do oscilador harmônico amortecido. Para inicio da aplicação de PINN's, foi utilizado como base o trabalho desenvolvido por Ben Moseley. No trabalho de Ben Moseley, (47), ele descreve como as PINN's incorporam o resíduo da equação diferencial nas funções de perda e em seguida, ele exemplifica como é feito utilizando o problema do oscilador harmônico amortecido.

O intuito de utilizar um código base foi para que pudessem ser testadas modificações, tais como: a variação de hiperparametros e como isso afeta a convergência, a variação da frequência de oscilação, a variação na arquitetura da rede, como as funções de perda se comportam e como a diferença entre a solução predita e a solução esperada variam ao longo das épocas.

O resultado de Ben Mosley, apresentado na figura 9, mostra a evolução temporal da posição, onde a curva em cinza representa a solução exata do oscilador harmônico, como mostrado na figura 6, a solução predita pela rede neural representada pela curva azul, os dados de treinamento representado pelos pontos em laranja e em verde os pontos em que a perda física é validada durante o treinamento do modelo, ou seja, são os locais no domínio escolhidos para que o modelo seja ajustado corretamente.

A arquitetura da rede neural de Ben Moseley, como ilustrada pela figura 9, é composta por, uma rede neural do tipo totalmente conectada (do inglês, *Fully Conected*), função de ativação do tipo tanh, 1 input, 1 output, 32 neurônios e 3 camadas escondidas (do inglês, *Hidden Layers*), número de épocas igual 1000, taxa de aprendizado igual a $1e^{-3}$, a função de perda total: $L_{total} = |(\ddot{x} + 2cw_0\dot{x} + w_0^2 x)|^2 + |(x(0) - x_0)|^2$, e o otimizador *Adam*. E para a solução do oscilador harmônico a frequência de oscilação utilizada é $w_0 = 20$ e a constante de amortecimento é $c = 2$.

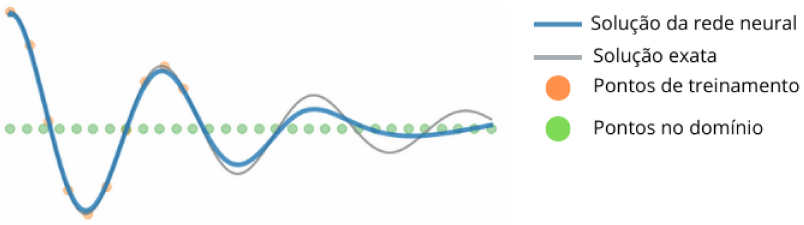

Figura 9 – Gráfico da evolução temporal da posição por Ben Moseley, onde a curva em cinza representa a solução exata do oscilador harmônico, a solução predita pela rede neural representada pela curva azul, os dados de treinamento representado pelos pontos em laranja e em verde os pontos são os locais no domínio escolhidos para que o modelo seja ajustado corretamente. Adaptado de:(48)

Baseado na descrição da rede neural de Ben Moseley, foi implementado um novo modelo, o nosso modelo de rede neurais artificiais informadas pela física, para testes de modificação e melhorias. Neste novo modelo, foram utilizadas as mesmas componentes da rede neural de Ben Moseley, a mesma arquitetura com os mesmos hiperparametros, a fim de comparar o modelo base com o novo modelo.

A figura 10 ilustra evolução temporal do novo modelo que foi implementado, composto pela mesma arquitetura e os mesmos parâmetros utilizados por Ben Moseley. É interessante notar que, a mudança na construção do modelo da rede neural afeta diretamente na solução predita. Dessa forma, foi necessário que os parâmetros se adequassem ao novo modelo para convergir corretamente.

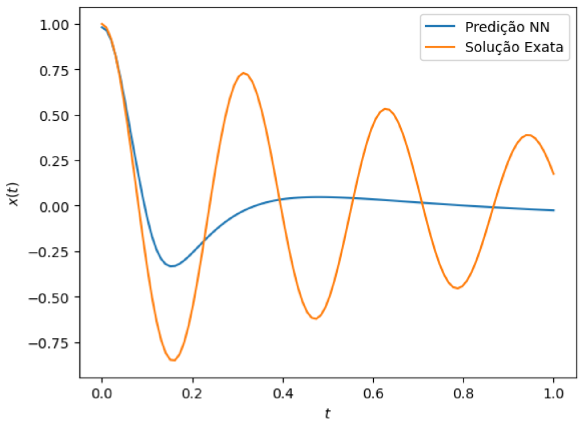

Figura 10 – Gráfico da evolução temporal da posição do nosso modelo implementado com os mesmos hiperparametros usados por Ben Moseley, o que difere aqui é que os dados de treinamento utilizados são as condições iniciais do sistema: x_0 e v_0 .

No novo modelo, algumas diferenças com o código base são: a criação de uma pasta chamada *pinn* contendo as principais bibliotecas usadas na execução do código que são: para importar as outras bibliotecas (*init.py*), para construir a animação (*animate.py*),

para definir a solução exata (*exato.py*), para definir a rede neural ($fcn.py$) e para definir a classe do oscilador harmônico (*oscilador.py*).

Outra diferença importante é que, os dados de treinamento não são definidos mais como pontos no domínio, como Ben Moseley usa, os dados de treinamento são definidos pelas condições iniciais do sistema. As condições iniciais são descritas pela equação 1.12 e pela equação 1.13, elas são usadas para inicializar o sistema físico, no caso, o sistema do oscilador harmônico no tempo inicial ($t = 0$). Assim, ao longo do treinamento, o modelo é ajustado para que as saídas estejam coerentes com as condições iniciais e de acordo com as leis da física que regem o problema do oscilador harmônico amortecido.

Além disso, é definido explicitamente os parâmetros do sistema do oscilador, tais como : a posição inicial (x_0), a velocidade inicial (v_0), o tempo máximo (t_{max}), a frequência (w_0) e a constante de amortecimento (c). A quantidade de dados de treinamento sendo definidas como um *batch*, os dados que são extraídos da solução exata, definidos como tempo exato (t_e), posição exata (x_e) e velocidade exata (v_e).

Devido a diferença na figura 9 e na figura 10, foram feitas algumas adequações para que o novo modelo tivesse uma convergência mais adequada. Essas adequações foram: mudança no número de épocas, ($epochs = 20000$) pois, não estava sendo suficiente para que a rede aprendesse o padrão. Também foi alterada a frequência para testarmos a convergência para um valor maior em comparação com o de Ben Moseley, então ($w_0 = 40$). O tamanho da rede neural também foi modificado para: o número de neurônios ($neurons = 5$) e as camadas para ($layers = 2$). A função de perda total definida como: $L_{total} = \frac{1}{w^4}|(\ddot{x} + 2cw_0\dot{x} + w_0^2x)|^2 + |(x(0) - x_0)|^2 + \frac{1}{w^2}|\ |(v(0) - v_0)|^2$, onde os termos: $\frac{1}{w^4}$ e $\frac{1}{w^2}$ são escolhidos para que todas as funções de perda possuem a mesma unidade.

Além disso, foram acrescentadas listas relacionadas as funções de perda para serem salvadas, e uma lista para serem salvados os valores de desvio (do inglês, *Devitation*), que é a diferença entre a solução exata e a solução predita pela rede. Com o intuito de que sejam plotados gráficos que relacionam as perdas com a evolução das épocas e o desvio com a evolução das épocas.

A figura 11 representa o gráfico da evolução temporal do modelo modificado, onde a curva em laranja representa a solução exata do oscilador harmônico, e a curva em azul representa a solução predita pela rede neural. Em comparação com a figura 10, que tem a mesma arquitetura de rede do código de Ben Moseley, a figura 11 com os hiperparametros ajustados apresenta uma convergência melhor, pois a curva que representa a solução predita pelo modelo se ajusta melhor a curva da solução exata.

Os resultados gerais obtidos a partir da adequação para o novo modelo, com os hiperparametros citados nos parágrafos anteriores, incluindo os gráficos que relacionam as funções de perdas e o desvio padrão com o número de épocas do modelo, estão ilustrados

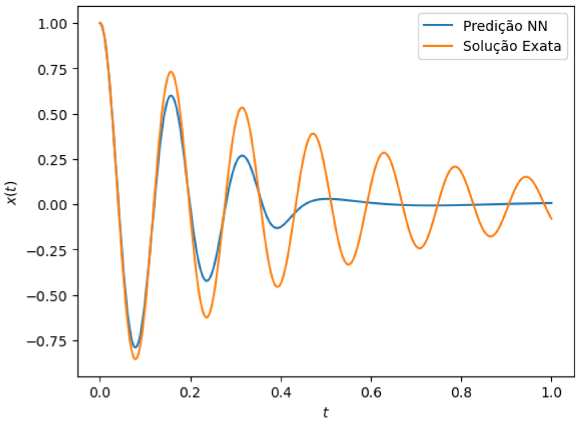

Figura 11 – Gráfico da evolução temporal da posição com as modificações dos hiperparametros da rede, tais como: mudança no tamanho da arquitetura da rede para 1 input, 1 output, 5 neurônios e 2 camadas, com os dados de treinamento sendo as condições iniciais de x_0 e v_0 e a função de perda acrescentando o resíduo da equação diferencial e as condições iniciais.

na figura 12.

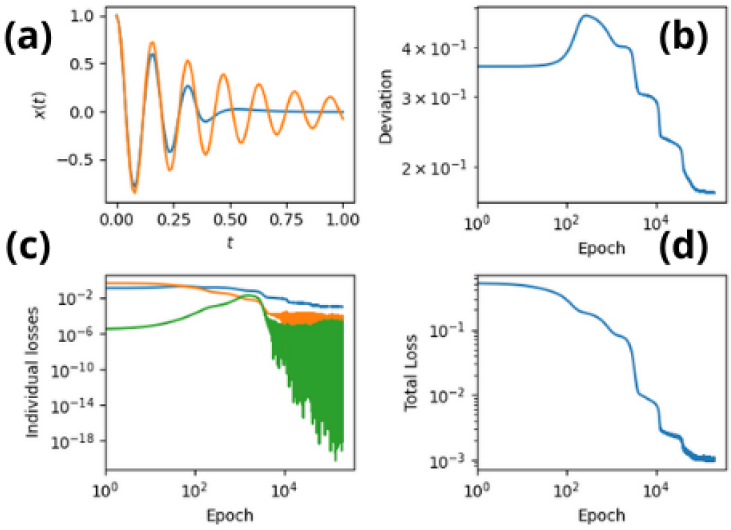

Figura 12 – Gráficos da evolução temporal (a), do desvio pelo número de épocas (b), das funções de perdas individuais pelo número de épocas (c) e do erro total pelo número de épocas (d).

Na figura 12. a) ilustram a evolução temporal da rede neural, onde a solução predita pela rede é representada pela curva azul e a curva laranja representa a solução exata do oscilador harmônico amortecido.

Os gráficos da 12. b) se referem ao número de épocas pelo desvio padrão, que é a diferença entre a solução exata pela solução predita. O desvio padrão calculado aqui se refere a diferença entre a solução exata do oscilador harmônico amortecido (x_e) menos a solução predita (x), é da forma:

$$deviation = \sqrt{\frac{\sum(x_e - x)^2}{N}} \tag{3.1}$$

Os gráficos da 12. c) são referentes a cada perda individual ao longo da evolução do número de épocas, a curva em azul representa L_1 descrita pela equação (2.1), que é a perda relacionada a equação diferencial, a curva em laranja representa L_2 descrita pela equação (2.2) que é o erro médio de x e a curva em verde representa L_3 descrita pela equação (2.3) que é o erro médio de v.

Os gráficos da 12. d) representa a perda total: $LT = L_1 + L_3 + L_3$ em função do número de épocas.

Mesmo com as modificações para melhorar a convergência, no sistema do oscilador harmônico amortecido, altas frequências de oscilação geram instabilidade no aprendizado do modelo, pois, o padrão nos dados de treinamento são dispersos. Com isso, a rede encontra dificuldades para generalização do problema.

3.1 Diferentes funções de ativação

As diferentes funções de ativação, em uma mesma rede neural, podem produzir diferentes predições, e a escolha de qual a melhor a ser utilizada depende, principalmente, do modelo criado.

Nesta seção, foram testadas diferentes funções de ativação, dentre elas: a tradicional função tanh, a função $Sigmoid$, a função $f(x) = x\tanh(x)$ e a função $f(x) = xSigmoid(x)$, com intuito de observar se acontece uma mudança significativa da convergência.

Para estes gráficos na arquitetura da rede, foram fixados hiperparametros tais quais, o número de épocas (do inglês, *epochs*) em 2×10^5, número de neurônios em 5, o número de camadas em 2, um batch size de 100 pontos, o passo de aprendizado (do inglês, *learning rate*) de 10^{-3} e a semente (do inglês, *seed*) fixada em (123).

Neste caso, é importante salientar que a semente é usada para gerar números aleatórios, como está fixada em (123) será gerado a mesma sequência de números aleatórios, isso garante que o mesmo resultado seja produzido em diferentes execuções do código.

Na figura 13 a) representa a evolução temporal de $x(t)$, a diferença está na função de ativação mencionada em cada gráfico. Comparando as figuras da coluna (a), o comportamento da curva em azul, que representa a predição do modelo, é perceptível que acontece um melhor ajuste da curva com a solução exata quando é utilizada a função do tipo $Sigmoid$.

Os gráficos da coluna (b), são referentes a diferença entre a solução real e a solução predita, que é o desvio padrão (*deviation*), pela a evolução do número de épocas de

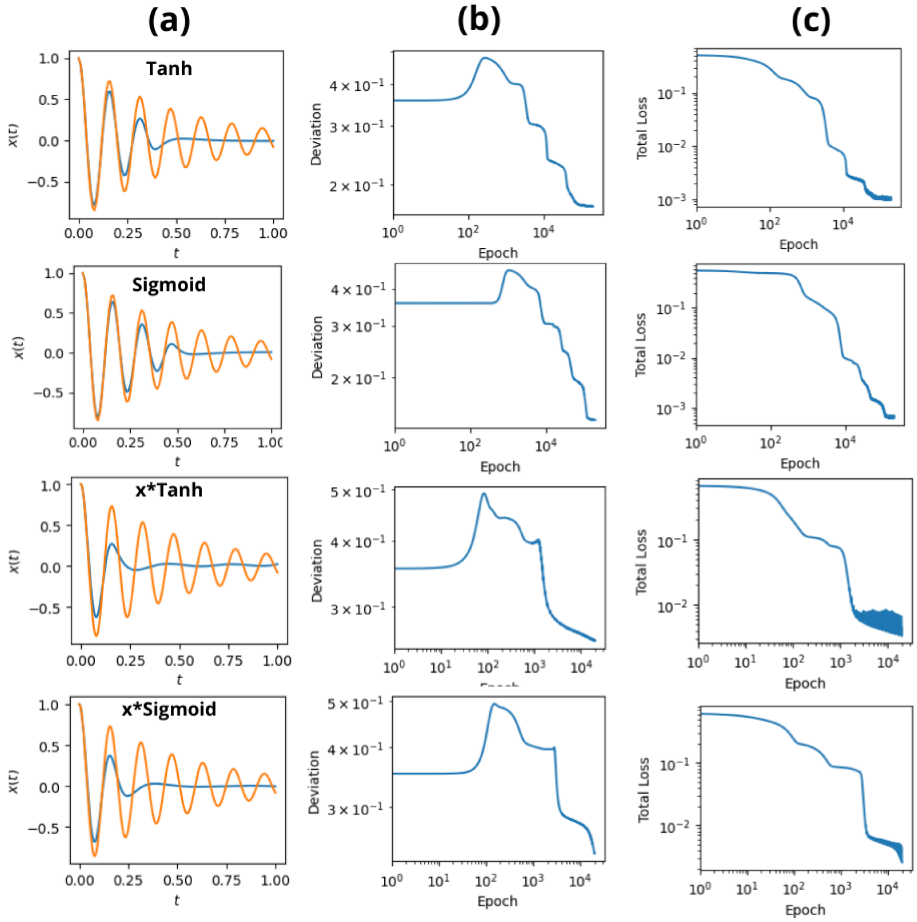

Figura 13 – Gráficos da evolução temporal (a), do desvio pelo número de épocas (b), e do erro total pelo número de épocas (c) em relação a diferentes funções de ativação.

treinamento ($epochs = 2 \times 10^5$). Neste gráfico, é percebe-se que, o aumento do número de épocas sucede a diminuição da diferença entre a solução da rede a solução real. A diminuição do desvio padrão é esperada, visto que, quanto mais perto a solução predita ficar da solução exata, melhor será a convergência da rede.

Os gráficos da coluna (c), são referentes a perda total, ou seja, $LT = L_1 + L_3 + L_3$, ao longo da evolução do número de épocas. Similar a analise para o desvio padrão pelo número de épocas, neste gráfico, também é visível a tendência de decrescimento ao longo do aumento do número de épocas.Isso ocorre porque a rede tenta reajustar os parâmetros livres (pesos e bias) a fim de melhorar os dados de treinamento, então, o decaimento nos gráficos demonstra que a rede está aprendendo a representar melhor os padrões nos dados e a fazer predições mais precisas.

3.2 Tamanho da NN

O tamanho da rede neural e o número de pontos de treinamento estão correlacionados e devem ser escolhidos cuidadosamente, porque, afetam diretamente a solução predita da rede. Sendo assim, a equação 2.5 é importante pois, relaciona o tamanho da rede, que são os neurônios e as camadas, com o número de parâmetros que devem ser reajustado durante o treinamento.

Dessa forma, o teste realizado consistia na seguinte arquitetura: tamanho da rede foi fixado em 5 neurônios e 2 camadas, totalizando 46 parâmetros. Na primeira linha da figura 14, o número de pontos de treinamento (*batch size*) foi fixado como sendo igual ao número de parâmetros.

Na segunda linha, o número de pontos de treinamento foi fixado como sendo 10 vezes o número de parâmetros e a última linha o número de pontos de treinamento foi fixado como sendo 100 vezes o número de parâmetros.

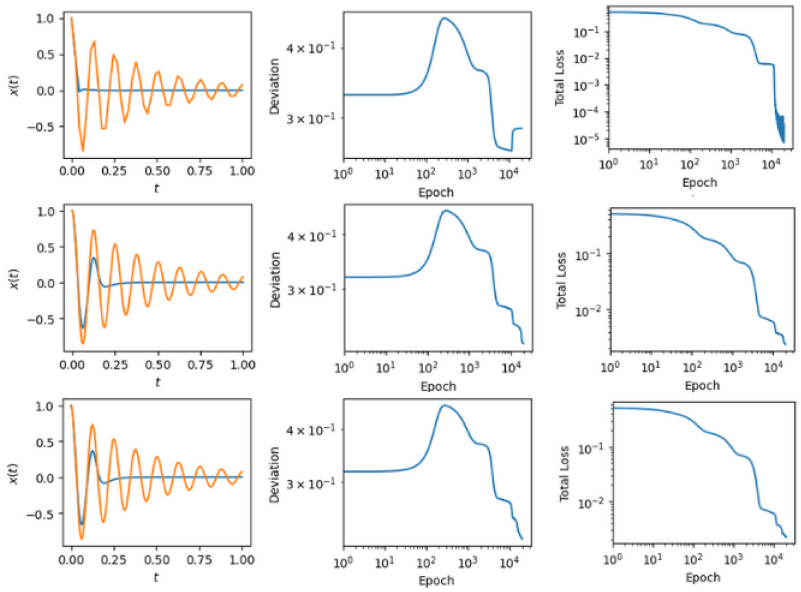

Figura 14 – Gráfico da evolução temporal da posição na primeira coluna, gráfico do desvio padrão pelo número de épocas na segunda coluna e gráfico da função de perda total pelo número de épocas, onde foram utilizados diferentes valores de batch, sendo eles: $batch = 46$ para primeira linha, $batch = 460$ para segunda linha e $batch = 4600$ para a terceira linha.

É perceptível como o número de pontos de treinamento deve ser maior que o número de parâmetros da rede, isso é ilustrado pela figura 14. O fato é que, a quantidade de parâmetros determinam o quão complexa a rede é. E por isso, os modelos que possuem uma grande quantidade de parâmetros tem uma capacidade maior de se ajustar e encontrar os padrões nos dados. Consequentemente, com o ajuste mais preciso a solução da rede se

torna mais precisa.

3.3 Epochs

Uma das principais etapas de treinamento é a escolha do número de épocas, é importante que seja adequado para o modelo resultar em uma boa predição. A importância dessa escolha reflete diretamente no treinamento pois, determina quantas vezes o modelo irá iterar sobre o conjunto de dados.

Como mencionado na seção 2.3, o número de épocas deve ser tal a evitar o *overfitting* e evitar o *underfitting*. Dessa forma, alguns testes foram realizados a fim de analisar como acontece a convergência da rede.

Na figura 15 ilustra como o número de épocas afeta diretamente no resultado. Foram realizados testes mudando o número de épocas de 2.000, 20.000 e 200.000

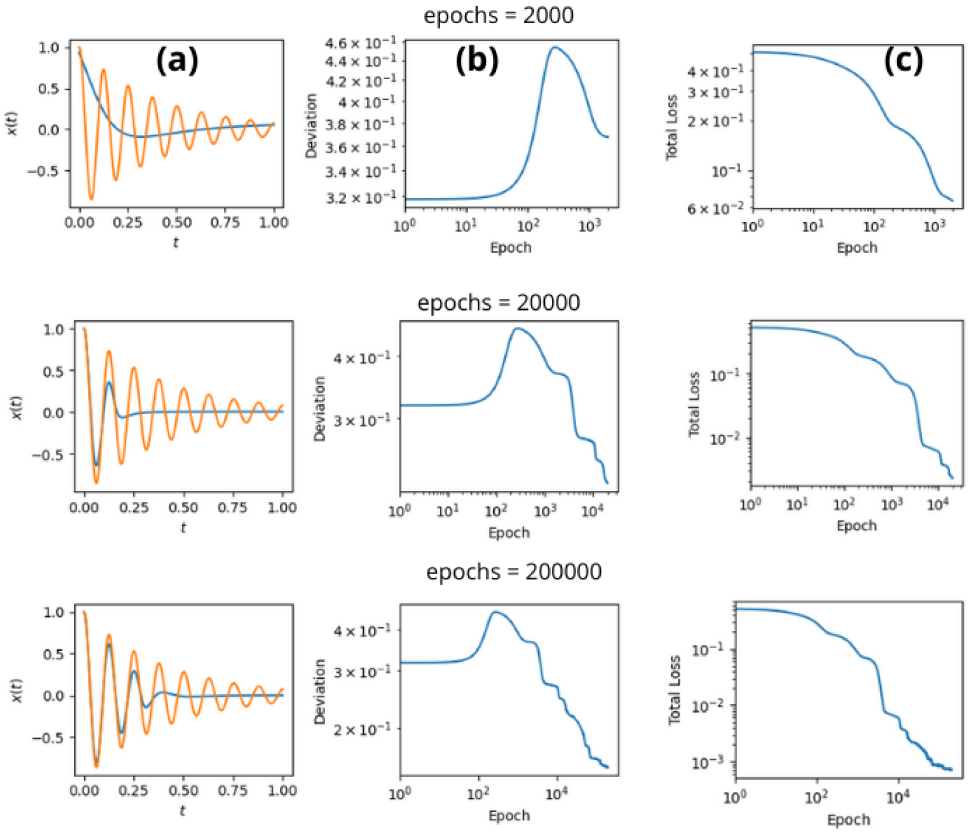

Figura 15 – Gráficos da evolução temporal (a), do desvio pelo número de épocas (b), e do erro total pelo número de épocas (c) em relação a diferentes números de épocas

Na figura 15 a coluna (a), representa a evolução temporal de $x(t)$ para diferentes valores de épocas, é possível notar que a curva em azul, descrita pela predição do modelo,

começa a se adequar a curva da solução exata em laranja, a partir do número de épocas igual a 20.000 e quando chega em 200.000 o ajuste do modelo melhora mais.

Além disso, a coluna (b) e a coluna (c) também tendem a resultados melhores quando o número de épocas é suficientemente grande. Em (b), o número de épocas pelo desvio tende a diminuir com o passar das épocas, isso porque a rede neural vai aprendendo a generalizar os padrões, porém, no primeiro gráfico, com o número de épocas igual a 2000 é notável que isso não acontece como o esperado. Da mesma forma em (c), a perda total no primeiro gráfico com o número de épocas igual a 2000, alcança a margem de 6×10^{-2} enquanto no último com o número de épocas igual 200000 alcança a margem de 10^{-3}.

Após analisar como o número de épocas influência na predição do modelo, é possível concluir que, neste caso, o número maior resultou em uma predição melhor. No entanto, é imprescindível levar em consideração os recursos computacionais quando o número de épocas é excessivamente grande, além do tempo de execução.

3.4 Convergência por intervalos

Durante o estudo de caso do oscilador harmônico amortecido, foi possível notar que, aumentar a frequência natural de oscilação (w_0), faz com que a convergência não apresente uma boa precisão. Assim, testando o modelo com valores de frequência maiores que 40 (valor de referência fixado), a generalização aprendida pela rede não funciona mais. Exemplificado pela figura 16, a solução predita pela rede neural, em azul, aprende bem no começo mas a generalização acaba convergindo para solução trivial.

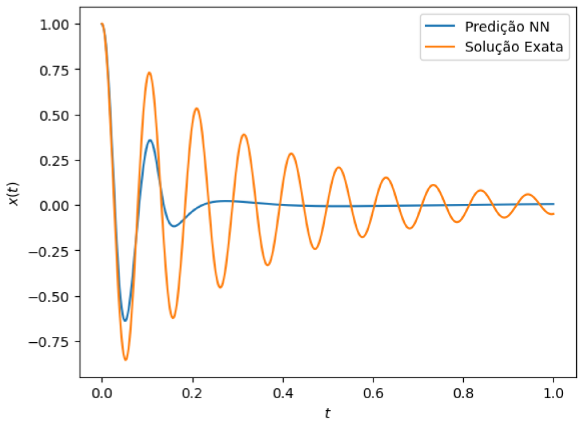

Figura 16 – Gráfico da evolução temporal da posição, onde a curva laranja representa a solução exata do oscilador harmônico amortecido e a curva em azul representa a solução predita pela rede, com frequência de oscilação $w_0 = 60$.

Devido a este obstáculo , a abordagem utilizada para melhorar a convergência da rede foi restringir a evolução temporal para pequenos δt, como proposto no trabalho realizado na referência (49). A evolução temporal é dividida em pequenos intervalos de

tempo, até o tempo máximo. O treinamento por pequenos intervalos, também conhecido como treinamento por sequência, consiste em restringir o intervalo evitando flutuações e generalizações que acontecem quando o aprendizado ocorre para tempos grandes.

O treinamento por sequência divide o processo de ajuste dos pesos da rede neural em passo menores e mais gerenciáveis, resultando em uma convergência mais suave para a solução adequada. Isso garante a estabilidade do treinamento fazendo com que o modelo possa aprender de forma consistente.

Este tipo de treinamento é útil, principalmente, em problemas com dados voláteis, dados que os padrões podem mudar ao longo do tempo. Então, o modelo consegue se adaptar de forma gradual aos diferentes padrões nos dados

Além disso, o treinamento por pequenos intervalos é vantajoso na hora do cálculo das derivadas. Em vez de calcular as derivadas a atualizar os pesos e bias para todos os dados em um grande intervalo de tempo, o modelo é treinado para fazer todos os cálculos em lotes menores, tornando o sistema computacionalmente mais vantajoso.

Então, foi implementado o treinamento por sequência, a arquitetura utilizada é a mesma que foi descrita na seção 3, com a função de ativação do tipo tanh e com a frequência ($w_0 = 60$). Comparando a figura 16 e a figura 17, ambas possuem os mesmos hiperparametros de rede, no entanto, a figura 17 tem o treinamento dividido em pequenos intervalos de tempo.

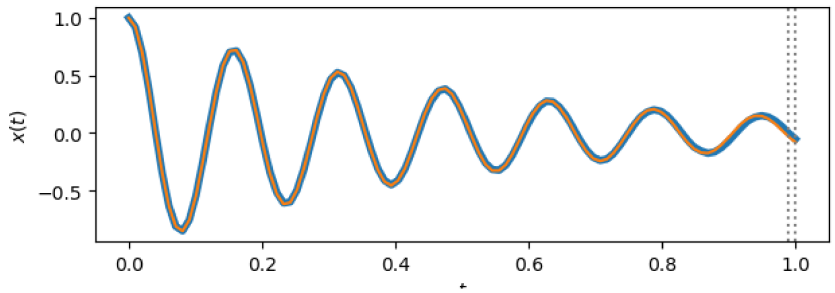

Figura 17 – Gráfico da evolução temporal da posição pelo método de aprendizado por sequência, onde é restringida a evolução temporal para pequenos intervalos de tempo.

De fato, é visualmente notável que, no caso do treinamento por sequência, há uma melhora na convergência da solução predita pela rede. Porém, durante os testes para este treinamento em sequência, foi observado que a convergência acontece de maneira correta quando aumentamos mais o número de épocas. Isto significa que a predição do modelo pelo aprendizado por sequência é bom, mas exige um número maior de épocas.

4 Conclusões e Perspectivas

Através deste trabalho, foi possível aprender e aplicar os conceitos de redes neurais. Também foi possível entender e testar as nuances das redes neurais informadas pela física. Utilizando como base o trabalho realizado por Ben Moseley em (47), os resultados gerados, de maneira geral, não apresentam um bom resultado.

Ao final do estudo exploratório das redes neurais pelo método das PINN's aplicados no oscilador harmônico amortecido, conclui-se que é um método promissor. No entanto, uma boa precisão foi obtida apenas no aprendizado por sequência descrito na seção 3.4, acresce que, o custo computacional foi maior neste caso, devido ao aumento no número de épocas.

Foi possível concluir também que não houve uma vantagem significativa na utilização de redes neurais artificiais em comparação a outros métodos computacionais tais como runge-kutta, em vista do tempo de execução, as redes neurais demoram mais tempo e não apresentam resultados significativamente bons.

As redes neurais informadas pela física possuem limitações, uma delas é lidar com sistemas de grandes oscilações, mas são uma boa abordagem para resolver equações diferenciais parciais de alta dimensionalidade, (34), (50), (51). E para a solução de equações diferenciais ordinárias de alta ordem, as redes neurais também apresentaram resultados efetivamente bons (52).No entanto, outros métodos computacionais, como Runge-Kutta são mais eficientes em termos de tempo execução

Assim como (39) nós concluímos que ainda são possíveis e necessárias inúmeras melhorias, no que diz respeitos as PINN's. Ainda há muito potencial a ser desenvolvido e explorado no treinamento ideal das redes neurais informadas pela física para resolver múltiplas equações e sistemas.

Todos os códigos podem ser encontrados em: https://gitlab.com/subgnano/pinn-oscillator

Referências

1 TURING, A. M. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX, n. 236, p. 433–460, 10 1950. ISSN 0026-4423. Disponível em: <https://doi.org/10.1093/mind/LIX.236.433>. Citado na página 15.

2 FLECK, L. et al. Redes neurais artificiais: Princípios básicos. *Revista Eletrônica Científica Inovação e Tecnologia*, v. 1, n. 13, p. 47–57, 2016. Citado 3 vezes nas páginas 15, 18 e 21.

3 RAUBER, T. W. Redes neurais artificiais. *Universidade Federal do Espírito Santo*, v. 29, 2005. Citado na página 15.

4 HAYKIN, S. O. *Neural Networks and Learning Machines*. 3. ed. Upper Saddle River, NJ: Pearson, 2008. Citado 4 vezes nas páginas 16, 17, 19 e 20.

5 MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, p. 115–133, 1943. Citado 2 vezes nas páginas 16 e 17.

6 MEDEIROS, R. Á. O. d. et al. Previsao de demanda no medio prazo utilizando redes neurais artificiais em sistemas de distribuicao de energia eletrica. Universidade Federal da Paraíba, 2016. Citado 2 vezes nas páginas 16 e 28.

7 WIKIPéDIA. *Função computável — Wikipédia, a enciclopédia livre*. 2023. [Online; accessed 2-setembro-2023]. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Fun%C3%A7%C3%A3o_comput%C3%A1vel&oldid=66526628>. Citado na página 16.

8 SEJNOWSKI, T. J.; TESAURO, G. The hebb rule for synaptic plasticity: algorithms and implementations. In: *Neural models of plasticity*. [S.l.]: Elsevier, 1989. p. 94–103. Citado na página 16.

9 HEBB, D. O. The first stage of perception: growth of the assembly. *The Organization of Behavior*, Wiley, New York, v. 4, n. 60, p. 78–60, 1949. Citado na página 16.

10 Wikipedia contributors. *Stochastic Neural Analog Reinforcement Calculator — Wikipedia, The Free Encyclopedia*. 2023. <https://en.wikipedia.org/w/index.php?title=Stochastic_Neural_Analog_Reinforcement_Calculator&oldid=1183172771>. [Online; accessed 15-December-2023]. Citado na página 17.

11 SILVA, F. da. *Previsão de inadimplência com Redes Neurais*. 2023. Disponível em: <https://analisemacro.com.br/econometria-e-machine-learning/previsao-de-inadimplencia-com-de-redes-neurais/>. Citado na página 19.

12 NASCIMENTO, P. *Aplicação de Sistemas Inteligentes no Controle de um Sistema Atuado por Material SMA*. 2015. Disponível em: <https://www.researchgate.net/figure/Figura-210-Aplicacao-do-algoritmo-de-retropropagacao_fig9_282330548>. Citado na página 23.

13 NABI, J. *Estimators, Loos Functions, Optimizers - Core of ML Algorithms*. 2019. Disponível em: <https://towardsdatascience.com/estimators-loss-functions-optimizers-core-of-ml-algorithms-d603f6b0161a>. Citado na página 25.

14 KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2017. Citado na página 25.

15 WIDROW, B.; LEHR, M. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, v. 78, n. 9, p. 1415–1442, 1990. Citado na página 25.

16 NIED, A. et al. On-line neural training algorithm with sliding mode control and adaptive learning rate. *Neurocomputing*, v. 70, n. 16, p. 2687–2691, 2007. ISSN 0925-2312. Neural Network Applications in Electrical Engineering Selected papers from the 3rd International Work-Conference on Artificial Neural Networks (IWANN 2005). Disponível em: <https://www.sciencedirect.com/science/article/pii/S092523120700094X>. Citado na página 26.

17 MACLAURIN, D.; DUVENAUD, D.; ADAMS, R. P. Autograd: Effortless gradients in numpy. In: *ICML 2015 AutoML workshop*. [S.l.: s.n.], 2015. v. 238, n. 5. Citado na página 26.

18 PASZKE, A. et al. Automatic differentiation in pytorch. In: *NIPS-W*. [S.l.: s.n.], 2017. Citado na página 26.

19 TEAM written by T. E. *PyTorch tutorial: a quick guide for new learnes*. 2021. Disponível em: <https://ai.plainenglish.io/pytorch-tutorial-a-quick-guide-for-new-learners-180957cb7214>. Citado na página 26.

20 A Gentle Introduction to torch.autograd. 2024. Disponível em: <https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html>. Citado na página 26.

21 THE Fundamentals of Autograd. 2021. Disponível em: <https://pytorch.org/tutorials/beginner/introyt/autogradyt_tutorial.html>. Citado 2 vezes nas páginas 26 e 27.

22 ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 27.

23 AMBRóSIO, P. E. *Redes neurais artificiais no apoio ao diagnóstico diferencial de lesões intersticiais pulmonares*. Tese (Doutorado) — Universidade de Sao Paulo, Agencia USP de Gestao da Informacao Academica (AGUIA), 2024. Disponível em: <http://dx.doi.org/10.11606/D.59.2002.tde-26102002-155559>. Citado na página 27.

24 CRUZ, T. N.; CRUZ, T. M.; SANTOS, W. P. Detection and classification of mammary lesions using artificial neural networks and morphological wavelets. *IEEE Latin America Transactions*, v. 16, n. 3, p. 926–932, 2018. Citado na página 27.

25 PASCANU, R. et al. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013. Citado na página 27.

26 IBM. *O que são redes neurais recorrentes*. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/recurrent-neural-networks>. Citado na página 27.

27 CECCON, D. *Os tipos de redes neurais*. 2020. Disponível em: <https://iaexpert.academy/2020/06/08/os-tipos-de-redes-neurais/>. Citado na página 28.

28 CHEN, M. et al. Deep feature learning for medical image analysis with convolutional autoencoder neural network. *IEEE Transactions on Big Data*, v. 7, n. 4, p. 750–758, 2021. Citado na página 28.

29 NIELSEN, M. A. *Neural Network and Deep Learning*. San Francisco, CA, USA: Determination Press, 2015. Disponível em: <http://neuralnetworksanddeeplearning.com/>. Citado na página 28.

30 ASSIS, C. A. S. et al. Restricted boltzmann machines for the prediction of trends in financial time series. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2018. p. 1–8. Citado na página 28.

31 LAKHANI, P.; SUNDARAM, B. Deep learning at chest radiography: Automated classification of pulmonary tuberculosis by using convolutional neural networks. *Radiology*, Radiological Society of North America (RSNA), v. 284, n. 2, p. 574–582, ago. 2017. ISSN 1527-1315. Disponível em: <http://dx.doi.org/10.1148/radiol.2017162326>. Citado na página 28.

32 YAMASHITA, R. et al. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, Springer Science and Business Media LLC, v. 9, n. 4, p. 611–629, jun. 2018. ISSN 1869-4101. Disponível em: <http://dx.doi.org/10.1007/s13244-018-0639-9>. Citado na página 28.

33 CHAPTER 4. Fully Connected Deep Networks. 2024. Disponível em: <https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html#:~:text=A%20fully%20connected%20neural%20network,follows%20in%20Figure%204%2D1.> Citado na página 29.

34 RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. 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. Disponível em: <https://doi.org/10.1016/j.jcp.2018.10.045>. Citado 3 vezes nas páginas 29, 30 e 49.

35 MOSELEY, B. *Physics-informed machine learning: from concepts to real-world applications*. Tese (Doutorado) — University of Oxford, 2022. Citado na página 29.

36 VILHENA, M. L. M. Uma breve introdução as equações diferenciais: Alguns tipos de equações diferenciais ordinárias. Citado na página 29.

37 BUSO, J.; FUMES, F. Um estudo sobre o uso de redes neurais para soluÇÃo de equaÇÕes diferenciais. 2023. Disponível em: <https://ocs.ifsp.edu.br/conict/xivconict/paper/view/9462/3588>. Citado na página 29.

38 OLIVEIRA, L. S. de et al. Dados de proveniencia para redes neurais guiadas pela fisica: o caso da equacao eikonal. In: SBC. *Anais do XXXVII Simpósio Brasileiro de Bancos de Dados*. [S.l.], 2022. p. 373–378. Citado na página 29.

39 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. ISSN 1573-7691. Disponível em: <http://dx.doi.org/10.1007/s10915-022-01939-z>. Citado 2 vezes nas páginas 30 e 49.

40 RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. E. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017. Citado na página 30.

41 SARTOR, M. L. *ANÁLISE DA APLICAÇÃO DE REDES NEURAIS FISICAMENTE INFORMADAS PARA A SOLUÇÃO DE EQUAÇÕES DIFERENCIAIS EM MECÂNICA DOS FLUIDOS*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2022. Citado na página 30.

42 MOSELEY, B.; MARKHAM, A.; NISSEN-MEYER, T. *Solving the wave equation with physics-informed deep learning*. 2020. Citado na página 30.

43 JIANG, X. et al. Physics-informed neural network for nonlinear dynamics in fiber optics. *Laser & Photonics Reviews*, Wiley, v. 16, n. 9, jul. 2022. ISSN 1863-8899. Disponível em: <http://dx.doi.org/10.1002/lpor.202100483>. Citado na página 31.

44 CAI, S. et al. Physics-informed neural networks (pinns) for fluid mechanics: a review. *Acta Mechanica Sinica*, Springer Science and Business Media LLC, v. 37, n. 12, p. 1727–1738, dez. 2021. ISSN 1614-3116. Disponível em: <http://dx.doi.org/10.1007/s10409-021-01148-1>. Citado na página 31.

45 FAN, D. et al. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 117, n. 42, p. 26091–26098, 2020. Citado na página 31.

46 NUSSENZVEIG H. MOYSéS (HERCH MOYSES), . *Curso de física básica, 2 : fluídos, oscilações e ondas, calor / Herch Moysés Nussenzveig* . -. 4.ed.. ed. São Paulo :: Edgard Blücher„ 2006[i.e.2002]. Inclui bibliografia e índice. Citado na página 31.

47 MOSELY, B. *So, what is physics-informed neural network*. 2021. Disponível em: <https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/>. Citado 3 vezes nas páginas 35, 39 e 49.

48 MOSELY, B. *Harmonic Oscillator - PINN*. 2021. Disponível em: <https://github.com/benmoseley/harmonic-oscillator-pinn>. Citado na página 40.

49 CHEN, H. et al. A critical evaluation of using physics-informed neural networks for simulating voltammetry: Strengths, weaknesses and best practices. *Journal of Electroanalytical Chemistry*, v. 925, p. 116918, 2022. ISSN 1572-6657. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1572665722009109>. Citado na página 47.

50 KHARAZMI, E.; ZHANG, Z.; KARNIADAKIS, G. E. *Variational Physics-Informed Neural Networks For Solving Partial Differential Equations*. 2019. Citado na página 49.

51 SIRIGNANO, J.; SPILIOPOULOS, K. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, v. 375, p. 1339–1364, 2018. ISSN 0021-9991. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999118305527>. Citado na página 49.

52 MALEK, A.; Shekari Beidokhti, R. Numerical solution for high order differential equations using a hybrid neural network—optimization method. *Applied Mathematics and Computation*, v. 183, n. 1, p. 260–271, 2006. ISSN 0096-3003. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0096300306005583>. Citado na página 49.