GERÊNCIA DE DADOS DE PROVENIÊNCIA PARA AS REDES NEURAIS GUIADAS PELA FÍSICA

Lyncoln Sousa de Oliveira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Marta Lima de Queirós Mattoso

Rio de Janeiro
Julho de 2023

GERÊNCIA DE DADOS DE PROVENIÊNCIA PARA AS REDES NEURAIS GUIADAS PELA FÍSICA

Lyncoln Sousa de Oliveira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Marta Lima de Queirós Mattoso

Aprovada por: Prof. Marta Lima de Queirós Mattoso
Prof. Adriano Maurício de Almeida Côrtes
Prof. Daniel Cardoso Moraes de Oliveira

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2023

Sousa de Oliveira, Lyncoln

Gerência de dados de proveniência para as Redes Neurais Guiadas pela Física/Lyncoln Sousa de Oliveira. – Rio de Janeiro: UFRJ/COPPE, 2023.

XIII, 65 p.: il.; 29, 7cm.

Orientador: Marta Lima de Queirós Mattoso

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2023.

Referências Bibliográficas: p. 60 – 65.

1. Proveniência. 2. Redes Neurais Guiadas Pela Física. 3. Humano no loop. I. Lima de Queirós Mattoso, Marta. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedico à minha família e amigos.

Agradecimentos

Primeiramente gostaria de agradecer à Deus, e também minha família pelo constante apoio e suporte. Em especial para minha mãe e meu pai, que apesar de não terem concluído se quer o ensino fundamental, sempre incentivaram meus irmãos e eu a estudar. Graças aos seus esforços, fomos aprovados e formados com sucesso em universidades públicas. Muito obrigado!

À Professora, e minha orientadora, Marta Mattoso. Obrigado por ter aceitado me orientar durante esses dois últimos anos. Apesar de terem sido anos de orientação remota, a Professora Marta sempre me deu o devido apoio e atenção para o desenvolvimento da minha pesquisa.

Ao Professor Daniel de Oliveira. Obrigado por ter me indicado para ser orientando da Professora Marta. Obrigado também por aceitar ser meu futuro orientador no doutorado.

Aos Professores Adriano Côrtes e Daniel de Oliveira por terem aceitado fazer parte da banca de avaliação desta dissertação.

Ao meu melhor amigo Daniel dos Santos, gostaria de expressar minha profunda gratidão pelos últimos dois anos de mestrado, nos quais ele esteve sempre presente para me auxiliar, seja nos estudos das disciplinas ou nos momentos de descontração.

Às minha eternas monitoras, e irmãs de orientação, Débora Pina e Liliane Kunstmann. Obrigado por me ajudarem durante esses dois anos de mestrado. A ajuda e o apoio de vocês foram fundamentais para o meu sucesso nas disciplinas. Obrigado de coração por estarem sempre de braços abertos.

À minha querida fiel companheira e namorada Mariana de Salles. Muito obrigado por ter me apoiado desde o momento em que me inscrevi no mestrado até agora no final. Sua presença foi essencial para preservar minha saúde psicológica e me manter equilibrado ao longo dessa jornada.

Por fim gostaria de agradecer as minhas queridas e amadas calopsitas, que me ajudaram a passar por momentos difíceis da minha vida, me alegrando com amor e inocência sempre que estavam por perto.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

GERÊNCIA DE DADOS DE PROVENIÊNCIA PARA AS REDES NEURAIS GUIADAS PELA FÍSICA

Lyncoln Sousa de Oliveira

Julho/2023

Orientador: Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

O aprendizado de máquina vem sendo usado cada vez mais em diferentes áreas de aplicação. Recentemente, as redes neurais vêm sendo adaptadas para realizar predições sobre soluções de equações diferenciais que representam fenômenos físicos. As redes neurais guiadas pela Física (PINNs), vêm revolucionando o uso de métodos numéricos na solução dessas equações por meio de modelos preditivos de redes neurais profundas. Apesar da complexidade de configuração e geração do modelo, uma vez treinado, o mesmo pode ser usado como um substitutivo do cálculo dos métodos numéricos, mostrando um ganho significativo em relação ao tempo de solução das equações. A incorporação da Física no treinamento desses modelos se dá por meio da modelagem e incorporação de componentes específicos na função de perda da rede neural. Tais componentes aumentam a complexidade na definição e análise das configurações de treinamento de modelos. Considerando a falta de apoio à análise comparativa da qualidade dos resultados dos modelos treinados, propomos a coleta e o uso de dados de proveniência, voltados à análise de modelos de PINNs. Analisamos a coleta de dados em ambiente computacional de alto desempenho, usando diferentes plataformas de treinamento de modelos de redes neurais, incluindo uma específica para PINNs. Experimentos com problemas de solução de equações diferencias parciais foram realizados evidenciando as contribuições desse registro específico de proveniência.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

PROVENANCE DATA MANAGEMENT FOR PHYSICS-INFORMED NEURAL NETWORKS

Lyncoln Sousa de Oliveira

July/2023

Advisor: Marta Lima de Queirós Mattoso

Department: Systems Engineering and Computer Science

Machine learning is being used increasingly in different application areas. Recently, neural networks have been adapted to predict solutions of differential equations that represent physical phenomena. Physics-Informed Neural Networks (PINNs) have revolutionized the use of numerical methods to solve these equations through predictive models of deep neural networks. Despite the complexity of configuring and generating the model, once trained, it can be used as a substitute for the calculation of numerical methods, showing a significant gain in relation to the time to solve the equations. The incorporation of Physics in the training of these models takes place through the modeling and incorporation of specific components in the loss function of the neural network. Such components increase the complexity of defining and analyzing model training configurations. Considering the lack of support for the comparative analysis of the quality of the results of the trained models, we propose the collection and use of provenance data, aimed at the analysis of PINNs models. We analyzed data collection in high-performance computational environments, using different neural network model training platforms, including one specific for PINNs. Experiments with problems of partial differential equations solutions were carried out showing the contributions of this specific record of provenance.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O Aprendizado de Máquina voltado ao treinamento de redes neurais profundas (DNNs) tem mostrado eficiência na resolução de uma variedade de problemas (KIDGER e LYONS, 2020). As DNNs são modelos de predição que têm como objetivo estimar uma função $\boldsymbol{f}(\boldsymbol{x})$ que mapeia elementos de entrada $\boldsymbol{x}$ para algum valor $\boldsymbol{y}$, de maneira que $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$. Dessa maneira, $\boldsymbol{\theta}$ é o conjunto de parâmetros que são adaptados durante o algoritmo treinamento de maneira a minimizar uma função que mede a perda de informação, comparando os dados de entrada com as estimativas do algoritmo (GOODFELLOW *et al.*, 2016). As DNNs Funcionam utilizando núcleos de processamento conhecidos como neurônios, que são responsáveis por processar os dados de entrada realizando cálculos baseados em regressão linear e produzir uma saída que pode ser processada em seguida por algum outro núcleo de processamento, até gerar o resultado da predição.

No processo de treinamento de uma DNN é necessário a escolha de hiperparâmetros, que são parâmetros definidos pelo usuário para o algoritmo da DNN antes do treinamento. Os hiperparâmetros definem como o processo de treinamento será realizado. Por exemplo, é necessário informar quantos neurônios irão compor a camada inicial da DNN, quantas serão as camadas intermediárias e quantos neurônios estarão presentes em cada uma delas. Também é necessário escolher alguma função de ativação entre os neurônios para gerar relações não lineares entre suas entradas e saídas, para assim aumentar o poder de predição do modelo para superfícies não lineares.

Depois de definida a configuração de hiperparâmetros, é realizado o treinamento que gera o modelo utilizando algum conjunto de dados de entrada. A qualidade da predição do modelo é medida por métricas de avaliação, que comparam o quão bem as estimativas do modelo performaram em comparação com os valores reais. A escolha dessas métricas dependem do tipo do problema. Como não há maneiras de definir previamente qual é a melhor configuração de hiperparâmetros que irá gerar o melhor modelo de predição para um problema, o usuário deve gerar diferentes con-

figurações que resultarão em diferentes modelos. Esse processo é conhecido como processo de geração de modelos. Ao final desse processo, é necessário realizar a escolha do melhor entre os modelos treinados. Essa escolha pode ser feita pelo usuário realizando a comparação dos resultados de uma, ou algumas, métricas de avaliação de todos os modelos. No geral, são testadas apenas algumas configurações, executar todas elas se torna inviável devido à grande cardinalidade do conjunto de possíveis configurações que são formadas pelas combinações dos diferentes hiperparâmetros. Por fim, o melhor modelo vai ser aquele gerado pela configuração de hiperparâmetros que proporcionou os melhores valores para essas métricas definidas.

LI *et al.* (2021) apresentam três abordagens que podem ajudar o usuário no processo de acompanhamento do treinamento e escolha de modelos, conforme é ilustrado na Figura 1.1. A primeira delas é a utilização de AutoML, representada pela Figura 1.1 (A). O AutoML é composto de algoritmos que buscam ajudar o usuário na obtenção da melhor configuração de hiperparâmetros para seu modelo de predição de maneira automática (KARMAKER *et al.*, 2021). O AutoML faz isso automatizando o processo de geração de configurações de hiperaparâmetros para os modelos, criando essas configurações baseadas em combinações de diferentes hiperparâmetros em um espaço de busca definido pelo usuário. Algoritmos de AutoML geralmente utilizam sistemas de otimização para navegar no espaço de busca dos hiperparâmetros, descartando combinações de hiperparâmetros que possuem maiores chances de não gerarem bons modelos. Uma vez executada as combinações de configurações, o AutoML seleciona aquela que resultou no melhor modelo utilizando como base alguma ou algumas métricas de avaliação definidas anteriormente pelo usuário. Sua aplicação pode ser custosa, pois dependendo da dimensão do problema, gerar diversas combinações de hiperparâmetros pode necessitar de bastante tempo e recursos computacionais, o que pode tornar seu uso inviável. Além disso, o usuário pode ter a conclusão durante a execução do experimento que a métrica de avaliação não é adequada, necessitando que a rotina do AutoML seja interrompida e executada novamente realizando as devidas alterações.

A segunda abordagem é a escolha interativa de modelo inserindo o humano no *loop*, representado pela Figura 1.1 (B). Nessa abordagem o usuário começa com algumas configuração de hiperparâmetros e obtém seus resultados. Com esses resultados e o conhecimento do domínio, o usuário pode realizar ajustes finos nos hiperparâmetros com objetivo de gerar melhores resultados em uma próxima execução. Esse processo é feito de maneira iterativa até a escolha do modelo feita pelo usuário com base em alguma métrica de avaliação, que pode ser alterada durante o desenvolvimento do experimento. A principal vantagem dessa abordagem é valorizar o conhecimento prévio do domínio pelo usuário, fornecendo controle sobre o processo de geração e escolha do modelo.

Já terceira abordagem é proposta por LI *et al.* (2021) e é chamada de paradigma intermitente de escolha de modelo com humano no *loop*, representada pela Figura 1.1(C), há uma junção das duas estratégias anteriores. O usuário define o espaço de busca das configurações da mesma maneira que a abordagem de AutoML necessita, porém agora além de visualizar os resultados durante a execução dos mesmos, também há a possibilidade de interromper, modificar ou adicionar novas configurações com base nos resultados anteriores. Assim as estratégias (B) e (C) tem potencial de valorizar o conhecimento prévio do usuário durante o processo de geração de modelos, facilitando a tarefa de escolha das melhores configurações e também reduzir o tempo e o esforço computacional em comparação com a estratégia (A). No geral, as três abordagens podem ser desenvolvidas e aplicadas independente do tipo de DNN.

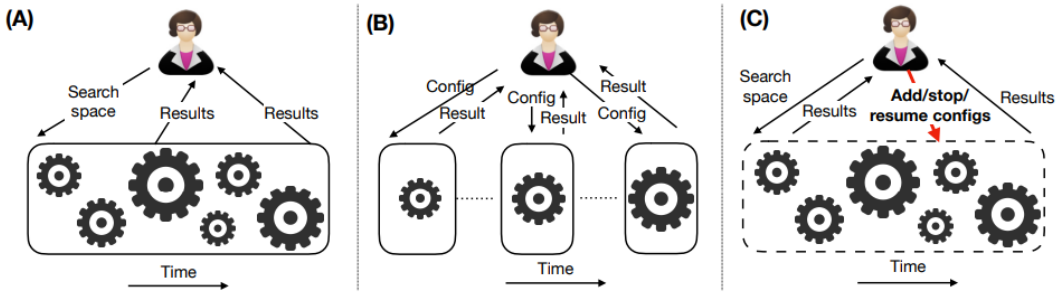

Figura 1.1: (A) Escolha de modelos por AutoML (B) Escolha interativa de modelo inserindo o humano no *loop* (C) Paradigma intermitente de escolha de modelo com humano no *loop* (LI *et al.*, 2021).

Uma das variações recentes de DNNs são as redes neurais guiadas pela Física (*Physics-Informed Neural Networks*-PINNs) que vêm revolucionando a solução de problemas governados por equações diferenciais parciais na ciência e na engenharia (RAISSI *et al.*, 2019). A incorporação da Física no treinamento de DNNs se dá por meio de novos componentes na função de perda. Algoritmos de DNNs buscam pelo modelo preditivo que minimiza a função de perda. Essa função de perda é geralmente calculada por um componente que leva consideração somente as estimativas produzidas pelo modelo e os dados de entrada. Em PINNs, além desse componente, os novos podem por exemplo, refletir a minimização do resíduo da equação diferencial e também incorporar aspectos das suas condições de contorno.

Assim, além de analisar o resultado da função de perda, é importante analisar cada componente de forma individual, pois cada um possui uma interpretação diferente para o problema. Para formar a função de perda, esses três componentes são combinados, por exemplo através de uma soma ponderada, onde cada um deles pode estar acompanhado de algum peso. Esses pesos podem ser escolhidos pelo cientista de maneira manual, o que estende a quantidade de hiperparâmetros da DNN, porém

há métodos que buscam realizar sua calibração de maneira automática durante o treinamento (BISCHOF e KRAUS, 2021; JIN *et al.*, 2021). De qualquer maneira, também pode ser importante para o cientista acompanhar a adequação desses pesos ou sua evolução, no caso de calibração automática.

Apoiar o humano no processo interativo de definição e escolha de modelos de DNNs em larga escala se faz necessário conforme KUMAR *et al.* (2021); VARTAK *et al.* (2016); WANG *et al.* (2019). Considerando as abordagens apresentadas anteriormente para ajudar o cientista nesse processo, temos que as abordagens (A) e (C) são limitadas para o caso de PINNs. Soluções de AutoML ainda são incipientes para PINNs, pois por exemplo, com esse método há dificuldade da compatibilização das informações da Física com o método automático de escolha do melhor modelo.

Com isso, nas PINNs o cientista ainda precisam acompanhar de perto e definir as configurações a serem avaliadas, considerando os hiperparâmetros de DNNs e os novos componentes das PINNs. É necessário também, independente da abordagens definidas na Figura 1.1, que o cientista defina alguma métrica adequada de avaliação para análise dos modelos gerados por cada configuração. Tais tarefas não são triviais para serem executadas e necessitam de um apoio de análise de configurações para o cientista que desenvolve as PINNs. Assim, nesse cenário, dentre as abordagens apresentadas por LI *et al.* (2021), a (B) parece ser a melhor opção a ser utilizada para apoiar o cientista no processo de escolhas de modelos de PINNs.

Para a adoção dessa abordagem, é necessário que o cientista faça a análise dos resultados parciais para realizar ajustes em novas rodadas. Realizamos uma busca de soluções que poderiam fornecer o apoio de análise de configurações exclusivamente às PINNs, porém não encontramos. Apesar da existência de ferramentas de apoio para análise de resultados de DNN, essas são limitadas, dificultando análise de modelos de maneira conjunta e também não foram feitas levando em considerações os novos elementos de PINNs, o que dificulta o processo de escolha de modelos.

Devido à falta desse apoio ao cientista de PINNs, esta dissertação propõe uma abordagem de análise de dados de PINNs baseada em proveniência. Essa abordagem fornece mecanismos de análise de resultados que facilitam a comparação dos resultados parciais, provenientes de cada configuração dos modelos treinados, através das métricas de avaliação e função de perda. Para isso, realizamos a persistência desses resultados de cada um dos modelos de maneira adequada, os interligando às suas configurações de hiperparâmetros. Nesse sentido, a adoção da proveniência desempenha um papel fundamental, pois representa o registro das informações sobre a origem, o histórico e as transformações sofridas pelos dados ao longo de seu ciclo de vida. Dentre as soluções de apoio à proveniência, se destaca a DNNProv (PINA *et al.*, 2020) que foi desenvolvida com o objetivo de realizar a implementação de proveniência para DNNs de maneira geral. Foi realizado uma modificação na bibli-

oteca da DNNProv para facilitar a gerência de dados de proveniência pela interação do cientista ao informar quais são os hiperparâmetros de entrada e as métricas de saída relacionadas ao treinamento das PINNs que deseja capturar e monitorar via proveniência.

A proveniência permite a reprodução dos resultados, o acompanhamento das alterações nos dados intermediários e a consulta de informações, auxiliando os cientistas a compreenderem melhor os resultados do seu experimento. No entanto, a grande quantidade de dados possíveis de serem gerados na execução de um experimento, mesmo em uma simples execução de um fluxo de trabalho científico, podem tornar a captura da proveniência dos dados uma tarefa complicada para os cientistas. Assim, é importante desenvolver mecanismos de geração de informações de proveniência que possam ajudar os usuários em seus experimentos, facilitando assim o processo de análise de suas execuções (NOURI *et al.*, 2021). Com isso, nesta dissertação abordamos o problema de escolha de modelos, explorando a organização dos resultados e seu uso para monitorar a evolução do treinamento de PINNs. Seguimos a ideia da seleção interativa de modelo inserindo o humano no *loop* apresentada pela Figura 1.1 (B), utilizando a proveniência para auxiliar nessa tarefa. Foram realizados experimentos que evidenciaram a flexibilidade da solução desenvolvida para capturar dados de proveniência para PINNs, e como esses dados ajudam o cientista na análise dos seus resultados.

Além da introdução, esta dissertação está organizada em outros 5 capítulos. O Capítulo 2 apresenta o que são modelos de redes neurais profundas, abordando os principais conceitos e posteriormente como a função de perda é expandida para aplicação das PINNs. Também apresenta a definição de proveniência dada pelo W3C PROV e como ela pode ser utilizada em DNNs e PINNs. O Capítulo 3 apresenta um mapeamento sistemático da literatura, buscando trabalhos que abordam a utilização de proveniência em PINNs. O Capítulo 4 apresenta a proposta desta dissertação, que aborda a análise de resultados de modelos de PINNs baseado em dados de proveniência. O Capítulo 5 apresenta os três experimentos realizados nesta dissertação, sendo dois deles problemas de análise e escolha de modelos em PINNs. Por fim, o Capítulo 6 apresenta a conclusão desta dissertação tomada com base nesses experimentos.

Capítulo 2

Definições de conceitos de DNNs, PINNs e proveniência

Este capítulo tem objetivo de apresentar conceitos teóricos utilizados nesta dissertação. Para isso, foi dividido em 4 seções. A Seção 1 apresenta como é o funcionamento das DNNs. A Seção 2 apresenta os principais conceitos sobre PINNs. A Seção 3 apresenta a definição de proveniência dada pelo W3C PROV. Por fim, a Seção 4 apresenta como a proveniência definida pelo W3C PROV pode ser utilizada no treinamento de modelos em DNNs e PINNs.

2.1 Deep Neural Networks (DNNs)

As redes neurais profundas (DNNs) são modelos de predição construídos por várias unidades de processamento que são agrupados em camadas. As camadas podem ser classificadas como inicial, onde os dados de entrada são inseridos no modelo; ocultas ou intermediarias que são onde os cálculos são realizados; e por fim a final que irá retornar a predição (GOODFELLOW *et al.*, 2016). A Figura 2.1 ilustra uma arquitetura de DNN.

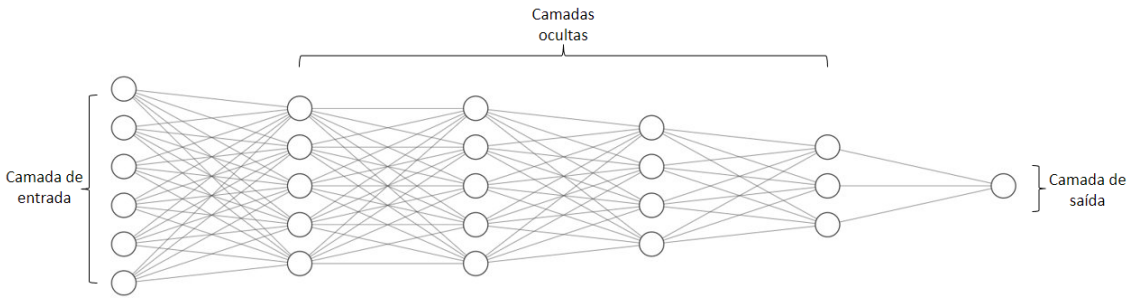

Figura 2.1: Modelo de rede neural profunda com camada inicial, quatro camadas ocultas e uma camada de saída.

O objetivo principal das DNNs é aproximar alguma função $\boldsymbol{f(x)}$ desconhecida que com dados de entrada $\boldsymbol{x}$, geraram as características de interesse $\boldsymbol{y}$. Para isso, os conjuntos de dados de entrada e suas características de interesse correspondentes são utilizados para aproximar $\boldsymbol{f}$, de maneira que $\boldsymbol{y} = \boldsymbol{f}^*(\boldsymbol{x}, \boldsymbol{\theta})$, onde $\boldsymbol{\theta}$ são os parâmetros treinados pelo algoritmo da DNN que resultaram em uma função $\boldsymbol{f}^*$ com a melhor aproximação da função real $\boldsymbol{f}$. Dessa forma, as DNNs podem ser utilizadas em tarefas de regressão, classificação, reconhecimento de padrões, entre outras (GOODFELLOW *et al.*, 2016).

No processo de treinamento do modelo, os parâmetros $\boldsymbol{\theta}$ são utilizados pelas unidades de processamento em conjunto com algum dado de entrada para gerar uma saída utilizando regressão linear. Como podemos observar pela Figura 2.1, as unidades de processamento realizam seus cálculos de maneira paralela em relação às camadas e suas saídas podem ser usadas como entrada de alguma outra até que o resultado da estimativa da característica de interesse seja apresentado pela camada de saída da rede.

Por realizar cálculos baseados em regressão linear, as unidades de processamento conseguem apenas representar superfícies lineares, o que torna o poder de predição do modelo ruim para a maioria das aplicações. Para contornar isso, são utilizadas funções de ativação nas saídas das unidades de processamento, tais funções são não-lineares e expandem a predição do modelo para superfícies mais complexas. São exemplos de função de ativação a ReLu, sigmoids, softmax e etc (LECUN *et al.*, 2015).

Uma vez na camada de saída, o valor estimado para a característica de interesse é comparado com o seu respectivo valor real por alguma função de perda, geralmente é utilizado o erro quadrático médio para problemas de regressão e entropia cruzada para classificação. Quanto o menor valor dessa função, melhor tende a ser a estimativa gerada pela DNN. Assim, é interessante que a função de perda seja minimizada, e isso pode ser feito utilizando métodos baseados em gradiente, que buscam encontrar os parâmetros $\boldsymbol{\theta}$ que minimizam a função de perda. Por fim, a retro-propagação do erro é realizada para atualizar os valores dos parâmetros $\boldsymbol{\theta}$ para as unidades de processamento da rede (GOODFELLOW *et al.*, 2016; LECUN *et al.*, 2015).

Embora DNNs tenham sido amplamente utilizadas em uma variedade de aplicações, esses modelos geralmente exigem uma grande quantidade de dados para o treinamento da rede. Dessa forma, em aplicações que envolvam Física, onde a coleta de dados pode ser difícil, demorada ou até mesmo impossível, há a necessidade de desenvolver novas abordagens para gerar soluções aceitáveis. Além de que em aplicações da Física, há a possibilidade do conhecimento prévio sobre os dados, por exemplo através de equações que descrevem o fenômeno Físico que regem o pro-

blema, tal informação não é aproveitada em DNNs comuns.

2.2 Physics–Informed Neural Networks (PINNs)

PINNs são tipos de DNNs utilizadas para gerar estimativas de soluções para problemas que envolvam fenômenos físicos. As PINNs levam em consideração a equação diferencial parcial (EDP) que caracteriza a Física do problema ao invés de tentar deduzir as soluções somente utilizando os dados de entrada. Para isso, PINNs podem resolver EDPs da forma mais geral, como é apresentado na Equação 2.1.

$$\begin{aligned} \mathcal{F}(\boldsymbol{u}(\boldsymbol{z});\gamma) &= \boldsymbol{f}(\boldsymbol{z}) \quad \boldsymbol{z} \in \Omega \\ \mathcal{B}(\boldsymbol{u}(\boldsymbol{z})) &= \boldsymbol{g}(\boldsymbol{z}) \quad \boldsymbol{z} \in \partial\Omega \end{aligned} \tag{2.1}$$

O domínio é definido por $\Omega \subset \mathbb{R}^d$ com contorno $\partial\Omega$. O vetor $\boldsymbol{z}$ informa as coordenadas espaço-temporais, $\boldsymbol{u}$ representa a solução desconhecida, γ é o conjunto de parâmetros relacionados com a física. A função $\boldsymbol{f}$ é responsável por identificar os dados do problema e $\mathcal{F}$ é o operador diferencial não linear. O operador $\mathcal{B}$ indica as condições inicias ou de contorno relacionados ao problema e $\boldsymbol{g}$ a função de contorno. A Equação 2.1 pode descrever problemas físicos tanto diretos quanto inversos. O objetivo de problemas diretos é encontrar a função $\boldsymbol{u}$ para todo $\boldsymbol{z}$ enquanto γ é o conjunto dos parâmetros específicos da física. Já para os problemas inversos, γ também é determinado a partir dos dados (CUOMO *et al.*, 2022).

No cenário das PINNs, $\boldsymbol{u}(\boldsymbol{z})$ é estimado utilizando auxílio de redes neurais, parametrizadas por um conjunto de parâmetros $\boldsymbol{\theta}$ de maneira que $\hat{\boldsymbol{u}}_\theta(\boldsymbol{z}) \approx \boldsymbol{u}(\boldsymbol{z})$. PINNs aproximam soluções de EDP treinando uma DNN que tem objetivo de minimizar uma função de perda que incorpora conhecimentos físicos do problema, como por exemplo, termos que refletem as condições de contorno, domínio e resídio da EDP em pontos selecionados do domínio.

PINNs são redes de aprendizado profundo que, dado um ponto de entrada no domínio de integração, produzem uma solução estimada nesse ponto da EDP após o treinamento. O algoritmo PINN é essencialmente uma técnica que encontra soluções de EDP convertendo o problema de resolver diretamente as equações governantes em um problema de otimização de função de perda. Funciona integrando o modelo matemático à rede e reforçando a função perda com um termo residual da equação governante, que atua como um termo penalizador para restringir o espaço de soluções aceitáveis (CUOMO *et al.*, 2022).

Dessa maneira, o funcionamento das PINNs pode ser entendido dividindo o processo por três principais blocos, que são: a rede neural, a rede informada pela Física e um mecanismo de *feedback* como é ilustrado na Figura 2.2.

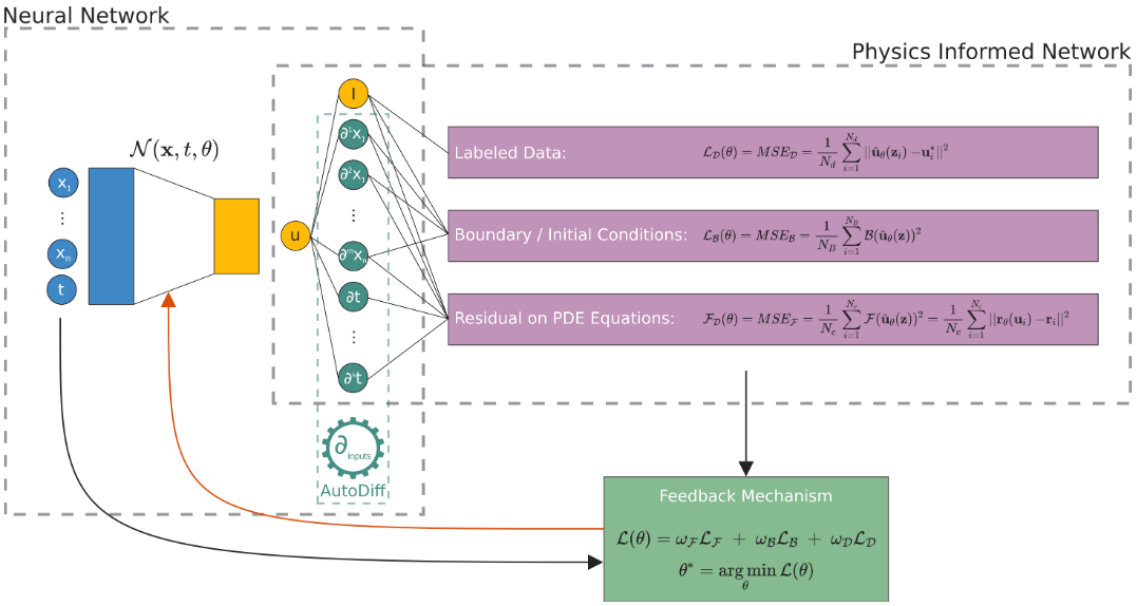

Figura 2.2: Blocos de construção de PINNs (CUOMO *et al.*, 2022).

O primeiro bloco da rede neural estima a solução $\hat{\boldsymbol{u}}_\theta$ que recebe como entrada o vetor $\boldsymbol{z}$ da Equação (2.1) e gera valores de $\boldsymbol{u}$. O segundo bloco calcula as derivadas para determinar os componentes da função de perda bem como os termos das condições iniciais e de contorno. Geralmente, esses dois blocos estão conectados por algoritmos de diferenciação automática (PASZKE *et al.*, 2017) que são usados para inserir equações físicas na rede neural durante a fase de treinamento. Assim, o mecanismo de *feedback*, que pode ser adotado por exemplo métodos baseados em gradiente, minimizam a função de perda apresentada pela equação 2.2 a fim de encontrar o melhor vetor de parâmetros $\boldsymbol{\theta}$ para estimar $\hat{\boldsymbol{u}}_\theta$ conforme mostra a equação 2.3.

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = w_R\mathcal{L}_{\mathcal{R}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{R}}) + w_{BC}\mathcal{L}_{\mathcal{BC}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{BC}}) + w_D\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{D}}) \tag{2.2}$$

$$\boldsymbol{\theta}^* = \arg\min_\theta(\mathcal{L}(\boldsymbol{\theta}; \mathcal{T})) \tag{2.3}$$

Como é apresentado em RAISSI *et al.* (2019); SILVA *et al.* (2021); SOUZA (2023), uma maneira eficiente de resolver problemas diretos e inversos associados a EDPs é por meio de PINNs. Para isso, as informações da Física regidas pelo problema são informadas para a PINN por meio da função de perda da rede, que pode ser representada por uma soma de componentes menores, que é mostrada na Equação (2.2), onde $\mathcal{T}_i, i = \mathcal{R}, \mathcal{BC}, \mathcal{D}$ são respectivamente os conjuntos de pontos de avaliação do resíduo, condições de contorno, dos dados relacionados ao problema.

A função de perda da rede $\mathcal{L}(\boldsymbol{\theta}; \mathcal{T})$ possui três componentes. O componente $\mathcal{L}_{\mathcal{R}}$

quantifica a qualidade de aproximação do resíduo da EDP, o $\mathcal{L}_{\mathcal{BC}}$ é responsável pela qualidade da aproximação das condições de contorno, por fim, o $\mathcal{L}_{\mathcal{D}}$ incorpora os dados. Cada componente pode estar associado a um peso w.

2.3 Dados de proveniência e o padrão W3C PROV

O *World Wide Web Consortium*[1] (W3C) é uma organização internacional que desenvolve padrões e diretrizes para a *World Wide Web*. Dentre os seus padrões, se encontra o W3C PROV[2], que é uma especificação que define um modelo de dados e uma linguagem para representar informações de proveniência. O W3C Prov define proveniência como "*informação sobre entidades, atividades e pessoas envolvidas na produção de um dado ou coisa, que pode ser usada para formar avaliações sobre sua qualidade, validade ou confiabilidade*" (MOREAU e GROTH, 2013). De acordo com essa definição, podemos entender que a proveniência representa o caminho de derivação de dados; que é a descrição dos procedimentos computacionais aplicados nos dados; associando os dados (entidades) com o algoritmos/programas (atividades) que transformam esses dados, além de cadastrar os agentes (pessoas, computadores) associado às entidades e atividades.

Em experimentos científicos, a proveniência desempenha um papel fundamental na interpretação e compreensão dos resultados. Ela nos permite examinar a sequência de passos que levaram a um determinado resultado experimental, proporcionando uma compreensão das cadeias de raciocínio utilizadas na sua obtenção. Além disso, a proveniência verifica se o experimento foi executado de forma adequada, identificando as entradas utilizadas e facilitando sua reprodução (FREIRE *et al.*, 2008).

Dentre as recomendações do W3C PROV se encontra o modelo de dados conhecido como PROV-DM (MOREAU *et al.*, 2013), que especifica como devem ser expressados os diferente elementos e relacionamentos de proveniência, conforme mostra o exemplo da Figura 2.3.

De acordo com GIL *et al.* (2013); MOREAU e GROTH (2013); MOREAU *et al.* (2013), os elementos e relacionamentos ilustrados na Figura 2.3 são definidos da seguinte maneira:

- **Entidades** são coisas físicas, digitais, conceituais ou de algum outro tipo. Podem ser tanto reais quanto imaginárias. São documentos, base de dados, arquivos, etc.

[1] https://www.w3.org/
[2] https://www.w3.org/TR/prov-overview/

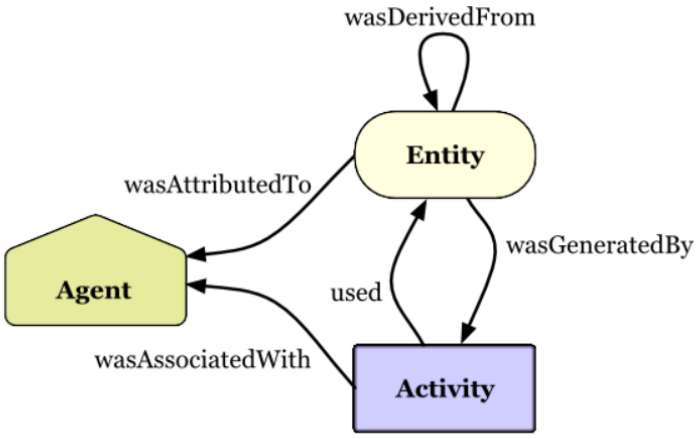

Figura 2.3: Elementos e relacionamentos do PROV-DM (GIL *et al.*, 2013).

- **Atividades** são eventos que ocorrem durante um período de tempo e agem em entidades, as fazendo existir ou/e mudar seus atributos para se tornarem outras entidades. Podem ser entendidas como aspectos dinâmicos, como por exemplo ações, processos, transformações, etc.

- **Agentes** são objetos que possuem de alguma forma responsabilidade por uma atividade que está ocorrendo, pela existência de uma entidade ou pela atividade de outro agente. São por exemplo cientistas responsáveis pelo experimento, instituições, computadores, etc.

Os possíveis relacionamentos de proveniência sobre esses elementos são mostrados na Tabela 2.1, onde cada relacionamento é descrito através de um conceito definido no PROV. Os nomes dos relacionamentos possuem sua forma verbal no passado para expressar a relação de um elemento com algum anterior.

Conceito PROV	Relacionamentos
Generation	WasGeneratedBy
Usage	Used
Communication	WasInformedBy
Derivation	WasDerivedFrom
Attribution	WasAttributedTo
Association	WasAssociatedWith
Delegation	ActedOnBehalfOf
Start	wasStartedBy
End	wasEndedBy

Tabela 2.1: Relacionamentos entre elementos do W3C PROV (MISSIER *et al.*, 2013).

Com o auxílio dos elementos e seus relacionamentos, é possível mapear todo um experimento utilizando proveniência e assim gerar seu caminho de derivação dos

dados. Assim posteriormente essa derivação pode ser usada para apoiar análises sobre qualidade, validade ou confiabilidade do experimento, também facilitando sua reprodutibilidade.

Para exemplificar o uso desse modelo de proveniência, observe a Figura 2.4. Essa ilustração retrata um exemplo de publicação de documentos de projeto no W3C *Consortium*[3] (MOREAU *et al.*, 2013). Projetos de trabalho são publicados regularmente para o W3C *Consortium* com o objetivo de refletir o trabalho realizado pelos grupos.

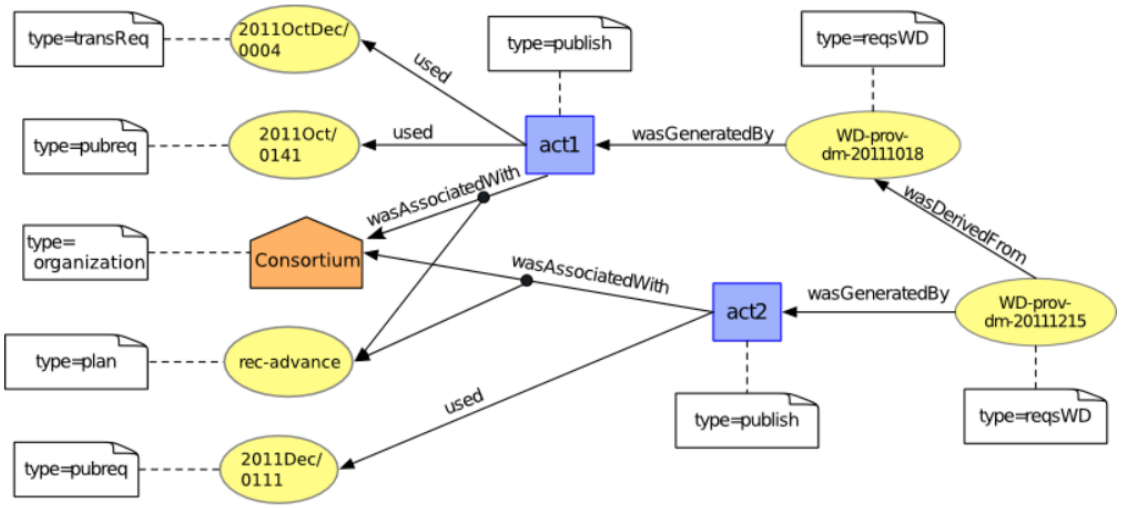

Figura 2.4: Proveniência de um documento (MOREAU *et al.*, 2013).

Nesse cenário duas entidades de versões de documentos estão envolvidas, *WD-prov-dm-20111018* que é o primeiro projeto de trabalho e *WD-prov-dm-20111215* o segundo. Ambos projetos foram publicados pelo agente *Consortium* do W3C. A atividade de publicação para o documento *WD-prov-dm-20111018* foi a atividade *act1* enquanto para o documento *WD-prov-dm-20111215* foi a *act2*. O segundo documento *WD-prov-dm-20111215* é derivado do primeiro documento *WD-prov-dm-20111018*. A atividade *act1* utilizou as entidade de requisição de publicação *2011Oct/0141* e requisição de transição *2011OctDec/0004*. A atividade *act2* usou a entidade de publicação *2011Dec/0111*. Ambos documentos foram publicados de acordo com a entidade de regras de processo *rec-advance*.

Dessa maneira, todo o processo de publicação de documentos de projeto foi mapeado utilizando proveniência. Assim, a proveniência além de ajudar a organização do processo, pode auxiliar na fácil obtenção de respostas sobre o mesmo, como por exemplo:

- Quais são as atividades associadas aos documentos dos projetos?

[3]https://www.w3.org/TR/prov-dm/

- Quais foram as entidades de publicação utilizadas pelas atividades?
- Qual é o caminho de derivação da entidade *WD-prov-dm-20111215* ?

Essas perguntas podem não ser facilmente respondidas caso não exista o auxílio da proveniência dada pelo modelo PROV-DM, o que pode acarretar dificuldade na garantia de qualidade, confiabilidade e reprodutibilidade do experimento. Os relacionamentos de proveniência tem papel importante para identificar a relação entre as atividades e entidades do processo de publicação de documentos de projeto. A PROV-DM pode ser usada em diversos cenários de aplicação, como por exemplo auxiliar um cientista no desenvolvimento no processo de seleção de modelos de DNNs e PINNs.

2.4 Proveniência para análise de resultados em DNNs e PINNs

O modelo de proveniência do W3C PROV também pode ser utilizado no contexto de treinamento de modelos de DNNs e PINNs. A utilização da W3C PROV pode ajudar o cientista na estruturação do processo de geração e escolha de modelos. Por exemplo, fazendo a associação de cada configuração de hiperparâmetros dos modelos aos resultados das suas métricas de avaliação.

O caminho de derivação dos dados pode incluir as atividades de treinamento, e adaptação de hiperparâmetros. As adaptações de hiperparâmetros podem ser utilizadas caso o cientista escolha mudar valores dos hiperparâmetros durante o treinamento. Por exemplo, é comum que os desenvolvedores de DNN utilizem adaptações para o hiperparâmetro de taxa de aprendizado, e no caso das PINNs, adaptações envolvendo os pesos dos componentes da função de perda. A proveniência pode ajudar na identificação dos valores dessas adaptações durante todo o treinamento do modelo, e como eles agiram nas modificações dos seus resultados.

Em DNN, a atividade do treinamento usa entidades de entrada como o conjunto de dados de treino e a configuração de hiperparâmetros, que são por exemplo: número de épocas, tamanho do *batch*, nome do otimizador, taxa de aprendizado, função de ativação, arquitetura da rede. A entidade de saída do treinamento conta com as métricas que avaliam o modelo, como a métrica de avaliação, a função de perda, o tempo decorrido e a data e hora do término da execução de cada intervalo de épocas. A atividade de adaptação usa o conjunto de dados gerados pelo treinamento e os hiperparâmetros que serão adaptados. A entidade gerada pela atividade de adaptação pode ser o identificador da adaptação, o novo valor para a taxa de aprendizado, a época, a data e hora da adaptação.

Em PINNs, além das entidades utilizadas em DNNs, também pode ser interessante incorporar informações da Física para o problema na entidade de entrada do treinamento, como por exemplo os pesos dos componentes da função de perda. As entidades de saída do treinamento também devem levar em conta os valores específicos para cada um dos componentes da função de perda. Desta forma, a cadeia de derivação entre treinamento e adaptação das PINNs fica representada por meio dos relacionamentos definidos no W3C PROV como é ilustrado na Figura 2.5.

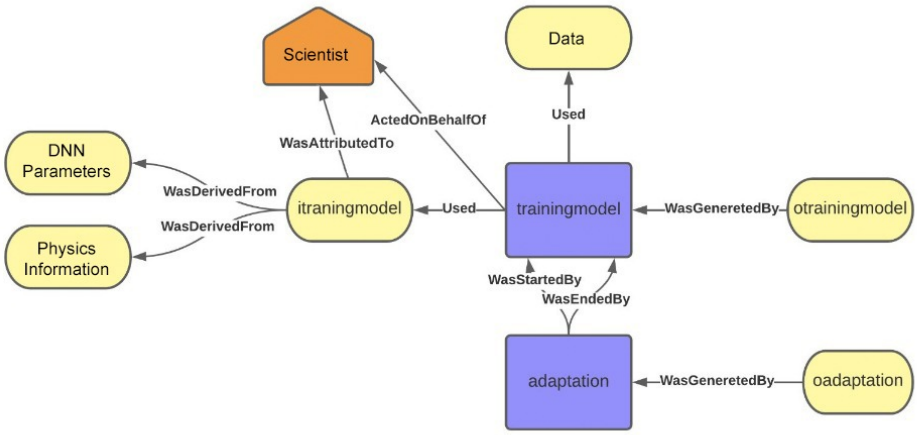

Figura 2.5: Proveniência do treinamento de PINNs definida por relacionamentos do W3C PROV.

Na Figura 2.5 são definidas seis entidades. As entidades "*DNN Parameters*" e "*Physics Informations*" representam respectivamente os hiperparâmetros da DNN e da Física. A entidade "*itrainingmodel*" representa as configurações de hiperparâmetros da PINN, enquanto a entidade ''*otrainingmodel*" as métricas de avaliação que são geradas durante o treinamento. A entidade "*oadaptation*" referencia aos valores adaptados dos hiperparâmetros durante o treinamento do modelo. A entidade "*data*" representa os dados de entrada do modelo. Em relação às atividades, temos "*trainingmodel*" que representa o treinamento da PINN, e "*adaptation*" as adaptações feitas durante esse treinamento. Por fim, temos o agente "*scientist*" que executa o algoritmo de treinamento e define a configuração de hiperparâmetros.

Capítulo 3

Mapeamento sistemático da literatura de proveniência em PINNs

A incorporação da Física no treinamento de PINNs apresenta desafios específicos que limitam o suporte oferecido pelos métodos clássicos de análise de modelos durante o treinamento. A adição de novos componentes à função de perda requer uma abordagem diferente, e a falta de suporte adequado para PINNs em ambientes convencionais de DNNs resulta em programação adicional por parte do usuário. Para apoiar o cientista na análise dos modelos a geração e captura de dados de proveniência é um fator importante em qualquer *workflow* científico, especialmente os que envolvem ML, para favorecer a reprodutibilidade e explicabilidade dos modelos.

Antes de propor uma solução para a análise de dados envolvidos na construção de modelos em PINNs, buscamos trabalhos na literatura com esse mesmo objetivo. Para isso, realizamos uma mapeamento sistemático com base em KITCHENHAM (2004); KITCHENHAM e CHARTERS (2007) com foco de analisar como os dados de proveniência podem auxiliar cientistas no desenvolvimento de PINNs. Pela ausência de artigos que abordam proveniência em PINNs, expandimos nossas questões de pesquisa para abranger também trabalhos que discutem como a proveniência pode apoiar o desenvolvimento de tarefas relacionadas a aprendizado de máquina (ML) e aprendizado de máquina científico (SciML). Dessa forma, podemos obter mais trabalhos e analisar se o conteúdo deles pode ser adaptado para a aplicação em PINNs. Para apoiar o desenvolvimento desse mapeamento usamos a plataforma web do Parsifal[1] que é uma ferramenta intuitiva que torna o processo de revisão mais eficiente e organizado.

O presente capítulo é dividido em outras 4 seções, a Seção 1 mostra quais foram as questões de pesquisa formuladas para esse mapeamento sistemático. A Seção 2 mostra como foram feitas as buscas de trabalhos. A Seção 3 apresenta quais são os

[1] https://parsif.al

critérios de seleção. A Seção 4 discute como os trabalhos selecionados respondem as questões de pesquisa levantadas.

3.1 Questões de Pesquisa

Nesta seção levantamos questões de pesquisa para investigar o papel da proveniência no ciclo de vida das aplicações ML, SciML e PINNs. Acompanhado de cada questão, descrevemos o objetivo que queremos atingir ao responde-las.

QP$_1$. Como proveniência pode contribuir no ciclo de vida das aplicações de ML, SciML e PINNs?

- Verificar se além de reprodutibilidade, confiança e princípios FAIR, a proveniência também pode ajudar o usuário no ciclo de vida das aplicações de ML, SciML e PINNs. Por exemplo, auxiliar o usuário nas tarefas relacionadas a configuração de hiperparâmetros, treinamento e seleção de modelos.

QP$_2$. Como capturar proveniência de aplicações em ML, SciML e PINNs?

- Estudar as abordagens e técnicas utilizadas. Verificar se há desafios específicos para ML, SciML e PINNs ou se eles são semelhantes aos encontrados em *workflows* científicos ou em outros domínios.

QP$_3$. Como proveniência é representada pelas soluções de captura de proveniência em aplicações ML, SciML e PINNs?

- Verificar se a representação das soluções encontradas permite seguir relacionamentos, como quais entidades foram usadas até a geração do modelo. Quando a solução representa apenas metadados como proveniência, esse tipo de consulta não é possível ou exige muito esforço por parte do usuário. Além disso, avaliar se a representação permite comparar modelos e se é compatível com o padrão W3C PROV.

QP$_4$. Quais soluções podem capturar proveniência em ambientes de HPC?

- Identificar se as soluções disponíveis têm potencial para serem executadas em ambientes de alto desempenho, como em super computadores HPC. O treinamento de modelos de PINNs geralmente são computacionalmente intensivos, pois envolvem a solução de equações diferenciais parciais que podem ser complexas.

QP$_5$. Quais soluções podem capturar proveniência independente do *framework*?

- Investigar quais dessas soluções necessitam ser executadas em *frameworks* de desenvolvimento específicos. Tal necessidade pode dificultar a adoção dessas soluções para usuários que já possuem familiaridade com outros ambientes de desenvolvimento. Em PINNs, existem *frameworks* específicos de desenvolvimento conhecidos, como por exemplo o NVIDIA Modulus (antiga SimNet) (HENNIGH *et al.*, 2021), DeepXDE (LU *et al.*, 2021) e SciANN (HAGHIGHAT e JUANES, 2021).

3.2 Busca

As buscas foram realizadas utilizando como base as seguintes palavras chaves: *deep learning*, *fine tuning*, *physics-informed neural networks*, *scientific machine learning*, *lifecycle*, *framework* e *provenance*. Para ampliar os resultados de busca, adicionamos palavras associadas mais conhecidas para cada palavra chave, com excessão de *provenance*, como é apresentado na Tabela 3.1:

Tabela 3.1: Sinônimos utilizados para as palavras chaves.

Palavra chave	Sinônimos
Deep learning	deep neural networks,machine learning
Fine tuning	tuning, automl, model selection, hyperparameter configuration, hyperparameter selection
Physics-informed neural networks	physics-based machine learning, physics-guided neural networks
Scientific machine learning	machine learning in science
Lifecycle	artifacts, experiments
Framework	tool, platform
Provenance	–

Dada essas informações, definimos a nossa primeira fonte de pesquisa. Construímos expressões de busca fazendo combinações desses termos e buscamos documentos em bibliotecas virtuais. Escolhemos o Google Acadêmico, devido sua abrangência, conseguindo buscar artigos de várias fontes. Também escolhemos a ACM Digital para complementar a pesquisa. Optamos por não fazer buscas em outras bibliotecas virtuais devido a quantidade de intersecções de trabalhos encontrados entre essas e o Google Acadêmico através das três expressões de busca. Pesquisamos pelo texto inteiro e sem filtro de data de publicação. Os resultados estão presentes na Tabela 3.2.

Tabela 3.2: Resultado das buscas nas bibliotecas.

Termos	Google Acadêmico	ACM Digital
fine tuning, physics-informed neural networks, provenance	34	0
deep learning, physics-informed neural networks, provenance	43	0
scientific machine learning, lifecycle, framework, provenance	73	2

Diferente das anteriores, a biblioteca virtual DBLP não possui acesso ao texto inteiro dos artigos, somente ao título e alguns metadados. Por esse motivo, as expressões formadas pelos termos da Tabela 3.2 não encontraram nenhum artigo.

Executamos uma busca mais genérica da forma "*provenance & (machine learning / deep learning)*" e assim obtivemos 26 trabalhos.

Como segunda fonte de pesquisa, consideramos artigos que conhecíamos e avaliamos relevantes pois abordam o uso da proveniência como caraterística principal no desenvolvimento da pesquisa em *workflows* científicos, esses são: SOUZA *et al.* (2019a), SOUZA *et al.* (2019b) e SOUZA *et al.* (2022). Definidos esses como artigos principais, realizamos *snowballing*, isso é, buscamos todos os trabalhos que citam algum desses três até o momento, o número de documentos que obtemos dessa maneira foram 50. A distribuição de documentos encontrados em cada biblioteca virtual é mostrado na Figura 3.1. Removendo a intersecção de trabalhos entre elas obtemos um total de 144 documentos.

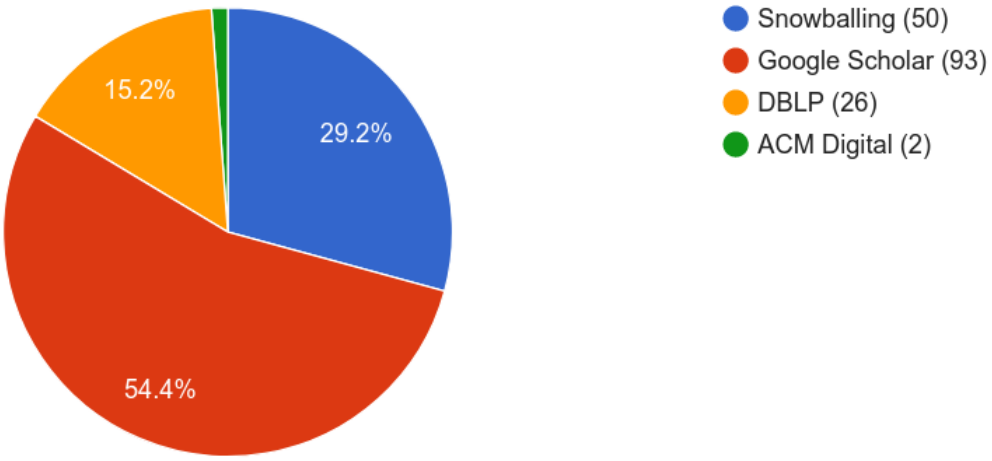

Figura 3.1: Distribuição de documentos encontrados por Biblioteca Virtual.

3.3 Critérios de seleção

De acordo com KITCHENHAM e CHARTERS (2007), os critérios de seleção do estudo se destinam a identificar os estudos que fornecem evidências diretas sobre as questões de pesquisa. Assim os critérios de inclusão (CI) e exclusão (CE) são construídos baseados nessas questões, devem ser interpretados de forma confiável para classificar os estudos corretamente.

3.3.1 Critérios de exclusão

CE$_1$. Trabalhos que apenas citam proveniência, sem apresentar a sua representação ou captura.

CE$_2$. Trabalhos que discutem proveniência em termos teóricos mas não aplicam.

CE$_3$. Trabalhos que usam proveniência em domínios diferentes de ciência da computação.

CE$_4$. Trabalhos não escritos em inglês.

CE$_5$. Trabalhos não revisados por pares.

3.3.2 Critérios de inclusão

CI$_1$. Trabalhos que abordam proveniência em workflows envolvendo DNN.

CI$_2$. Trabalhos que abordam proveniência em workflows envolvendo PINNs.

CI$_3$. Trabalhos que abordam proveniência em workflows envolvendo SciML.

CI$_4$. Trabalhos que abordam proveniência no contexto de HPC.

3.4 Discussões

Após realizadas a seleção dos trabalhos, realizamos discussões de como esses podem responder as questões de pesquisa levantadas no início deste capítulo. Ao total foram filtrados 23 trabalhos. A Tabela 3.3 mostra a distribuições dos trabalhos filtrados por critério de inclusão, onde um trabalho pode ser aceito por mais de um CI. Observamos que apesar das nossas questões de pesquisa estarem associadas a proveniência em PINNs, somente um trabalho aborda esse tema.

Tabela 3.3: Distribuição de trabalhos aceitos por critério de inclusão.

Critério	Quantidade
CI_1	19
CI_2	1
CI_3	14
CI_4	19

3.4.1 QP$_1$: Como proveniência pode contribuir no ciclo de vida das aplicações de ML, SciML e PINNs?

Aplicações científicas têm potencial de lidar com grande volumes de dados. Para auxiliar cientistas durante seus experimentos, soluções automatizadas podem ser

adotadas para persistir dados; que podem ser resultados gerados pelo experimento; e seus metadados associados; que são informações que fornecem detalhes sobre os dados, como sua origem, contexto, formato; com objetivo de ajudar análise para tomada de decisão (WOZNIAK *et al.*, 2022). Nesse contexto, a adoção de proveniência é interessante, pois além de persistir os dados, também mapeia o seu caminho de derivação, assim tornando os dados do experimento localizáveis, acessíveis, interoperáveis e reutilizáveis (FAIR). Além disso, proveniência é fator chave para garantir reprodutibilidade em experimentos de ML, que é uma característica importante e desejada para experimentos científicos (BAKER, 2016; HUTSON, 2018; MISSIER, 2016; SAMUEL e KÖNIG-RIES, 2021).

Proveniência é especialmente interessante em cenários de aprendizado de máquina, onde modelos treinados podem ser artefatos importantes para analisar os resultados do experimento e garantir reprodutibilidade (FAGNAN *et al.*, 2019). Por exemplo, durante a construção de redes neurais, os cientistas precisam analisar como diferentes configurações de hiperparâmetros afetam a qualidade do modelo treinado. Com o auxílio da proveniência os cientistas podem comparar conjuntamente os resultados de vários modelos e selecionar o melhor para suas necessidades e ao final possuirá o caminho de derivação dos dados que chegou nesse resultado (KUNSTMANN *et al.*, 2021).

Apesar de existir ferramentas que buscam automatizar o processo de escolha relacionadas a modelos, tais decisões ainda são em grande parte determinadas pela experiência do cientista, assim a proveniência dos dados é um fator crítico para possibilitar a análise e adaptação em tempo real desses modelos. Nesse sentido, ferramentas de suporte de captura de dados com proveniência devem ser desenvolvidas de maneira conjunta entre os especialistas de proveniência e cientistas do domínio do experimento, para que todos os dados importantes da aplicação sejam devidamente persistidos e suas relações sejam mapeadas (STEVENS *et al.*, 2020).

3.4.2 QP$_2$: Como capturar proveniência de aplicações em ML, SciML e PINNs?

Em *workflows* científicos, as aplicações tem potencial de gerar uma grande variedade de dados e metadados. Com isso, a proveniência muitas vezes é empregada com alta granularidade, ou seja, capturando todas as informações geradas pelo experimento (HAN *et al.*, 2022). Essa abordagem pode não ser a mais adequada, já que informações desnecessárias são persistidas, o que pode levar a problemas de armazenamento e performance para o experimento.

Em aplicações que envolvam a construção de modelos de ML, é crucial que os modelos treinados e conceitos específicos, como hiperparâmetros, sejam integrados no modelo de proveniência. Para o processo de geração de modelos, é necessário considerar todas as versões dos modelos de ML produzidos durante o processo in-

terativo de geração de modelos (WOZNIAK *et al.*, 2022). Essa tarefa é essencial para a escolha do modelo, já que, com base nos resultados de modelos anteriores, o cientista pode realizar ajustes nos hiperparâmetros com o objetivo de obter modelos treinado melhores (KUNSTMANN *et al.*, 2021).

Em cenários científicos é comum que aplicações envolvam múltiplos *workflows*. Nessas situações, o ciclo de vida das aplicações pode exigir transformações nos dados brutos para treinamento de modelos, o que envolve a adoção de fluxos de trabalho com diversos algoritmos, dados, ferramentas de processamento e armazenamento de dados. Essas atividades são realizadas por equipes multidisciplinares, compostas por cientistas do domínio, cientistas e engenheiros de computação, e especialistas em ML. A proveniência dos dados é importante para rastrear, reproduzir, avaliar e entender os dados e os processos de transformação, assim, a captura desses dados pode ser feita dentro de cada *workflow*, assim como nas integrações dos dados entre os diferentes *workflows* (SOUZA *et al.*, 2019a,b, 2022).

3.4.3 QP_3: Como proveniência é representada pelas soluções de captura de proveniência em aplicações ML, SciML e PINNs?

Em HAN *et al.* (2022), os autores apresentam o PROV-IO, que é um framework de proveniência para dados científicos em sistemas HPC que se concentra em operações de entrada e saída de dados (I/O). Para modelar os dados de proveniência, os autores estendem o padrão W3C PROV adicionando subclasses que podem descrever dados, operações de I/O e ambientes de execução.

No trabalho de SILVA *et al.* (2018), os autores apresentam a solução de captura de proveniência para *dataflows* chamada DfAnalyzer[2], que é uma ferramenta que permite monitorar, depurar e analisar fluxos de dados gerados por experimentos científicos em tempo real, persistindo os dados utilizando o SGBD MonetDB[3], que é voltado para o armazenamento de dados científicos. A DfAnalyzer também segue um modelo de dados compatível com o W3C PROV (MISSIER *et al.*, 2013) para representar relacionamentos entre conjuntos de dados manipulados por modelos computacionais, sendo compatível com diferentes domínios de aplicação científica.

O uso da DfAnalyzer é empregado na ferramenta Keras-Prov[4], que é focada em aplicações que envolvam construção de redes neurais utilizando o *framework* Keras (KUNSTMANN *et al.*, 2021; PINA *et al.*, 2021); e DNNProv, que utiliza o mesmo *back-end*, é agnóstica ao *framework* de execução (PINA *et al.*, 2021). Em SILVA *et al.* (2021) a DNNProv foi utilizada em cenário de PINNs, onde além de persistir dados comuns de DNNs, os autores estenderam para capturar os valores da função de perda e seus componentes ligados a Física. Por usarem a DfAnalyzer, possibilitam

[2]https://github.com/ElsevierSoftwareX/SOFTX_2019_102
[3]https://www.monetdb.org/
[4]https://github.com/dbpina/keras-prov

aos cientistas analisarem os dados do treinamento durante a sua execução utilizando o SGBD MonetDB.

Em SOUZA *et al.* (2019a), SOUZA *et al.* (2019b) os autores apresentam a ferramenta chamada de ProvLake. É uma ferramenta de captura de dados de proveniência que utiliza o padrão W3C PROV, desenvolvida para ser utilizada em múltiplos *workflows* científicos. A ProvLake Realiza a persistência dos dados para cada um desses *workflows* e também para a ligação entre eles.

No trabalho de WOZNIAK *et al.* (2022), é apresentada a ferramenta Braid-DB, cujo objetivo principal é capturar a evolução dos modelos de ML produzidos no processo interativo de geração dos modelos. Embora os autores não citem um modelo de proveniência específico, eles projetaram estruturas de dados e implementaram funcionalidades para permitir a geração automática de identificadores, a coleta de metadados descritivos e a construção de registros de proveniência. Essas funcionalidades permitem a reprodutibilidade, análise *post-mortem* e auditoria dos cálculos, facilitando a compreensão e reprodução dos resultados obtidos. Apesar de ser uma ferramenta que também é voltada para SciML, ela não provê apoio para PINNs.

Em MUSTAFA *et al.* (2023) os autores apresentam a ferramenta MLProvCodeGen, que é uma solução de proveniência desenvolvida para apoiar os cientista na captura de proveniência em experimentos em formato de *notebook*. Tal formato é diferente dos *scripts* convencionais, o experimento pode ser dividido em células individuais e podem ser executados interativamente pelo usuário. A ferramenta é uma extensão para o JupyterLab, onde através de uma interface visual permite, por exemplo, o cientista selecionar opções como quais informações serão capturadas e tipo de problema, podendo escolher entre classificação de imagens ou multi-classes usando DNN. A partir da seleção, a ferramenta gera código de proveniência em conjunto com um *notebook* que é criado a partir de um template associado ao tipo de problema. O MLProvCodeGen utiliza a biblioteca prov do Python, que faz uso de proveniência do W3C PROV, representando cada célula como entidade, atividade como a execução do seu conteúdo que gera uma nova entidade relacionado as dados de saída.

No geral, se a proveniência não é adotada, os dados das transformações precisam ser rastreados manualmente pelo cientista. Essa tarefa é complexa e pode ser demorada, além de ser propensa a erros (SCHLEGEL e SATTLER, 2023). Isso pode ser prejudicial tanto do ponto de vista científico, afetando a reprodutibilidade dos resultados, quanto do ponto de vista de negócios, já que os usuários podem não se sentir confiantes em aplicar um modelo treinado, mesmo que com melhor desempenho, se não entenderem completamente as transformações realizadas ao longo do ciclo de vida do modelo (SOUZA *et al.*, 2019a).

3.4.4 QP$_4$: Quais soluções podem capturar proveniência em ambientes de HPC?

Os experimentos de PROV-IO foram executados em supercomputador com 64 nós de processadores Intel Xeon Haswell com até 4.096 núcleos. O Braid-DB teve seus experimentos executados em um computador com processador intel i7-8700, porém os autores citam que estão desenvolvendo a ferramenta para ser executada também em ambientes de alto desempenho. O MLProvCodeGen por ser uma ferramenta para arquivos do tipo *notebook*, que são arquivos interativos, não tem objetivo de ser executado em HPC, já que seria necessário permanecer com o *notebook* aberto durante toda execução de um experimento, que pode demorar dias. A Keras-Prov teve seu experimento executado no super computador Lobo Carneiro do Núcleo de Atendimento a Computação de Alto Desempenho COPPE/UFRJ, que possui 504 nós com processadores Intel Xeon Haswell. A DNNProv foi utilizada no super computador GRID5000, usando computação híbrida de GPU-CPU. Já o ProvLake em um dos seus experimentos foi executado em dois super computadores. O primeiro denominado *learning cluster* que tem 396 nós intel e power8, onde cada possui 24 a 48 núcleos de processamento; o segundo denominado *data processing cluster*, que tem 12 nós com dois processadores intel com 40 núcleos.

3.4.5 QP$_5$: Quais soluções podem capturar proveniência independente do *framework*?

Nas soluções encontradas o ProvLake, DNNProv, PROV-IO e Braid-DB não mencionam a necessidade de *frameworks* de execução específicos nos seus experimentos. Keras-Prov foi construída para ser uma ferramenta de proveniência para o *framework* Keras do TensorFlow. Já o MLProvCodeGen, que é uma extensão para JupyterLab, possui como opção de execução o *framework* PyTorch.

Capítulo 4

Apoio de dados de proveniência em PINNs

Este capítulo está dividido em 2 seções. A Seção 1 define a proposta desta dissertação, que aborda como a utilização de proveniência pode ajudar o cientista de PINNs. A Seção 2 mostra o quanto a captura de proveniência pode impactar o tempo de processamento das PINNs.

4.1 Proposta de captura de dados de proveniência em PINNs

A solução proposta desta dissertação tem como objetivo principal avaliar como a utilização da proveniência pode apoiar os cientistas desenvolvedores de PINNs no problema de geração e escolha de modelos independente do *framework* de treinamento. Tomamos como base a proposta (B) apresentada por LI *et al.* (2021) chamada de "Escolha interativa de modelo inserindo o humano no *loop*", ilustrada pela da Figura 1.1 (B). Assim, o conhecimento do cientista será valorizado o colocando com controle durante o processo de geração e escolha de modelos de PINNs.

A fim de atingir esse objetivo, avaliamos ferramentas apresentadas no Capítulo 3, do nosso mapeamento sistemático, que poderiam ter potencial para essa tarefa. Avaliamos principalmente os seus potenciais de uso em ambientes de super computadores, já que o treinamento de modelos de PINNs geralmente são computacionalmente intensivos, pois podem envolver a solução de equações diferenciais parciais complexas; extensibilidade em relação a modelagem de componentes específicos de PINNs e ao uso independente do *framework* de treinamento.

Nesse sentido, as ferramentas ProvLake e DNNProv se destacaram. Comparando essas duas ferramentas, a ProvLake tem como objetivo a captura de dados de proveniência entre diferentes *worflows* cietíficos de maneira geral. Nesta dissertação

estamos focados na geração e escolha de modelos em PINNs, e com isso, a ferramenta DNNProv se torna mais adequada, pois é especializada na geração de dados de proveniência para tarefas envolvendo DNNs. Apesar da biblioteca DNNProv fornecer extensibilidade em relação à captura de dados de proveniência, instrumentar essa captura com os dos dados de entrada e saída do treinamento pode ser uma tarefa difícil para o cientista.

Atualmente a DNNProv fornece por padrão um modelo pronto da definição dos dados para serem capturados, tanto para os dados das configurações dos hiperparâmetros, tanto para as métricas de saída geradas durante o treinamento. Porém essa definição conta com hiperparâmetros e métricas predefinidas de DNNs comuns. Não é possível fazer alterações nesses padrões, sendo assim necessário que o cientista faça novas transformações para capturar seus dados de interesse utilizando outros recursos da DNNProv. Realizar essas transformações não é uma tarefa trivial e pode fazer o cientista não optar pela utilização dessa solução. Nesse sentido, realizamos uma modificação[1] na biblioteca DNNProv para que seja mais fácil e intuitivo para o cientista informar quais são os hiperparâmetros de entrada e as métricas de saída relacionadas ao treinamento que deseja capturar.

Para utilizar a DNNProv, primeiramente é necessário instalar algumas dependências de *softwares*, que são especificadas pelo repositório da autora[2]. Essas dependências são:

- Java - Linguagem utilizada para o funcionamento da DfAnalyzer
- MonetDB - SGBD utilizado para persistir os dados
- DfAnalyzer (SILVA *et al.*, 2018) - RESTful api para geração de dados de proveniência
- dfa-lib-python (CAMPOS, 2018) - Integração da DfAnalyzer para a linguagem Python

Para seu funcionamento, a DNNProv necessita que o sistema DfAnalyzer seja executado anteriormente para fazer a comunicação entre o *script* de treinamento através da dfa-lib-python e o SGBD MonetDB para persistir os dados utilizando o padrão W3C PROV, representado pela Figura 4.1. Nessa figura, além dos elementos e relacionamentos de proveniência definidos pelo padrão W3C PROV, também é possível observar os dados referentes aos hiperparâmetros e métricas de saída de DNNs que são persistidos em cada entidade pelos retângulos brancos interligados à esses elementos.

[1] https://github.com/lyncoln/dnnprov
[2] https://github.com/dbpina/dnnprov

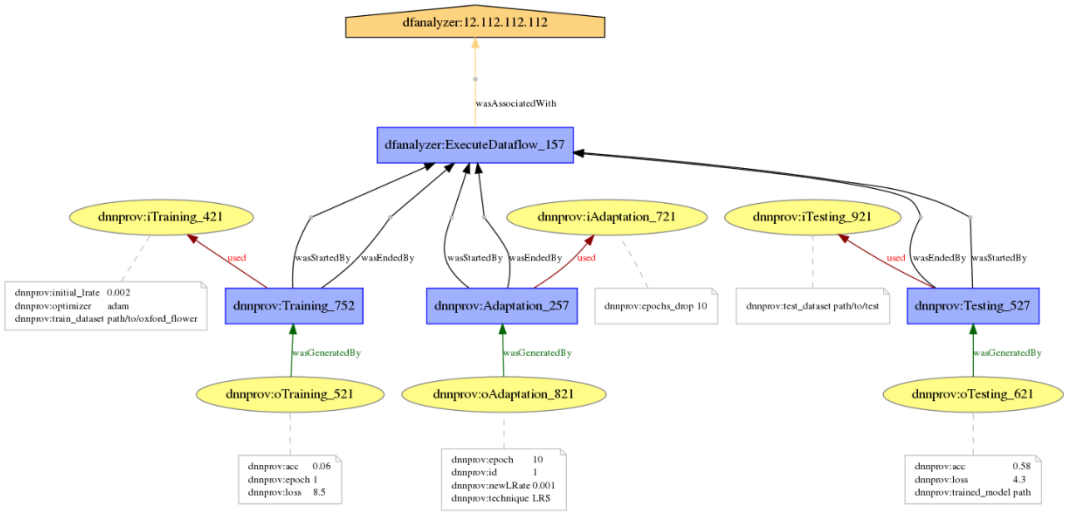

Figura 4.1: Exemplo de uma instância de execução da DNNProv com W3C PROV.

A modificação realizada nesta dissertação é referente à customização dos dados a serem persistidos nas entidades *iTraining*, que é responsável pelos hiperparâmetros de entrada do modelo; e *oTraining*, que é responsável pelas métricas de saída, que no caso das PINNs pode ser por exemplo a função de perda e seus componentes, e a métrica de avaliação. Pela Figura 4.1 a DNNProv já espera que os dados persistidos nessas entidades sejam fixos e relacionados a DNNs comuns. A nossa modificação permite que o cientista informe quais dados devem ser persistidos em cada uma dessas entidades.

Utilizando a nossa modificação, é necessário que durante a construção do *script* de treinamento das PINNs o cientista defina manualmente utilizando a DNNProv quais serão os dados de entrada, referente à configuração de hiperparâmetros; e saída do experimento, referente aos valores das métricas de avaliação que deseja capturar durante o treinamento. A modificação realizada na DNNProv foi referente à passagem dessas informações. Um exemplo de código da definição dos dados do experimento da Seção 5.3 é ilustrado abaixo.

```
df = Dataflow(dataflow_tag = "PINN_eikonal",
['STR_OPTIMIZER_NAME',
'NUM_LEARNING_RATE',
'NUM_NUM_EPOCHS',
'NUM_BATCH_SIZE',
'STR_ACTTT',
'STR_ACTVEL',
'STR_ARQ_TT',
'STR_ARQ_VEL',
'NUM_WEIGHT_LB',
```

```
'NUM_WEIGHT_LR',
'NUM_WEIGHT_LD'],
['NUM_epoch',
'NUM_time_elapsed',
'NUM_LOSS',
'NUM_LB',
'NUM_LR',
'NUM_LD',
'NUM_R2'])
df.save()
```

O objeto df criado pela função *Dataflow* é responsável pela inicialização da DNNProv e também pela construção das entidades de entrada e saída relacionadas a atividade de treinamento. O primeiro argumento dessa função é a *dataflow_tag*, que é a *tag* de identificação que será utilizada para identificar o experimento no SGBD. O segundo é uma lista que irá compor as colunas da entidade *itrainingmodel*, que representa os dados de entrada. Já o terceiro argumento é uma lista que irá compor as colunas da entidade *otrainingmodel*, que representa os dados de saída. A nomeação dos componentes devem respeitar a regra de começar com "STR_" caso a informação tenha formato de texto e "NUM_" numérica. Definido o objeto df, é necessário utilizar o método *save()* para que as tabelas referentes às entidades e ligações sejam geradas no MonetDB. A modificação feita pela DNNProv se limita somente na criação do objeto df, fora isso, a utilização da DNNProv é a mesma definida pelas sua desenvolvedora.

Já com o objeto df criado e salvo, é necessário que o cientista passe as informações manualmente para preencher as entidades. Primeiro vamos levar em consideração a entidade *itrainingmodel* que é preenchida uma única vez por execução. O seu preenchimento é feito da seguinte maneira:

```
exec_tag = dataflow_tag + datetime.now()
t1 = Task(1, dataflow_tag = dataflow_tag, exec_tag = exec_tag, "TrainingModel")
tf1_input = DataSet("iTrainingModel",
[Element([opt,
starter_learning_rate,
epochs,
batch_size,
act_tt,
act_vel,
layers_tt,
layers_vel,
weight_lb,
weight_lr,
weight_ld])])
t1.add_dataset(tf1_input)
```

```
16 t1.begin()
```

Na linha 2, o usuário deve criar um objeto com a função *Task()*, que faz a referência de uma atividade. Para isso, é necessário passar a primeiramente a numeração da atividade, atividades ligadas a entidades de treinamento possuem numeração 1; depois a *dataflow_tag* utilizada no passo anterior; em seguida uma *tag* de execução para diferenciar as execuções do mesmo dataflow; e por último o nome da atividade, que deve ser passada como "TrainingModel" caso a atividade seja de treinamento, e "Adaptation" caso seja de adaptação.

Na linha 3 são definidas as informações que serão persistidas na entidade de entrada *iTrainingModel* referentes aos hiperparâmetros. O cientista deve criar um objeto com a função *DataSet()*, informando como primeiro argumento o nome da entidade, e em seguida a lista de informações que serão atribuídas à essa entidade. Essa lista deve ser passada utilizando a função *Element()*, e deve respeitar a ordem dos elementos criados pela primeira lista do objeto df. Após essas definições, o objeto *DataSet* é conectado com a tarefa da atividade através do método *add_dataset()* e por fim é inicializado com o método *begin()* para que a atividade esteja disponível para informar os dados de saída.

Agora com os dados de entrada persistidos, o cientista pode passar os dados de saída que serão persistidos pela entidade *oTrainingModel*. Para isso, devemos seguir um procedimento parecido com o anterior, porém ao invés de ser definido uma única vez, para os dados de saída é necessário executar a seguinte linha de código para cada passo do treinamento o qual o usuário queira persistir as informações de saída.

```
1  tf1_output = DataSet("oTrainingModel",
2  [Element([epoch,
3  time_elapsed,
4  loss_value,
5  Lb,
6  Lr,
7  Ld,
8  r2])])
9  t1.add_dataset(tf1_output)
10 t1.save()
```

Do mesmo jeito feito para o *DataSet* com a entidade *iTrainingModel*, os dados presentes na lista devem estar na mesma ordem que foram criados no objeto df. Ao final o cientista deve utilizar o método *save()* para persistir os dados no SGBD.

Na definição do objeto df, uma entidade de adaptação é criada de maneira automática para capturar as adaptações da taxa de aprendizado ao decorrer do treinamento. Essa entidade pode ser usada caso o cientista necessite dessa informação no

seu experimento, ou pode não preenche-la caso contrário. Para persistência desses dados, o cientista deve seguir os mesmos passos anteriores com pequenas modificações. Na função *Task()* o cientista precisa passar através do argumento *dependency* qual é a atividade que essa adaptação está envolvida, que no caso é a de treinamento. O código abaixo é um exemplo de implementação.

```
1 t2 = Task(2, dataflow_tag, exec_tag, "Adaptation", dependency=t1)
2 t2.begin()
3
4 tf2_output = DataSet("oAdaptation", [Element([new_learning_rate_value, epoch])])
5 t2.add_dataset(tf2_output)
6 t2.save()
```

As linhas 1 e 2 são referentes à atividade adaptação e deve ser definida uma única vez, enquanto as linhas 4 a 6 salvam os dados na entidade *oAdaptation* no SGBD e devem ser executados todas as vezes em que o cientista desejar persistir tal informação.

Além das transformações relacionadas as atividades de treinamento e adaptação da taxa de aprendizado, o cientista pode criar novas transformações dependendo das necessidades do seu experimento. O código a seguir fornece um exemplo de criação de uma transformação qualquer. Após criada, as maneiras de iniciar e adicionar dados são análogas ao que já foi apresentado.

```
new_transformation= Transformation("SomeTransformation")
new_transformation_input = Set("iSomeTransformation", SetType.INPUT,
    [Attribute("some_parameter1", AttributeType.NUMERIC),
    Attribute("some_parameter2", AttributeType.NUMERIC)])
new_transformation_output = Set("oSomeTransformation", SetType.OUTPUT,
    [Attribute("some_output", AttributeType.TEXT)])
new_transformation.set_sets([new_transformation_input,
                             new_transformation_output])
df.add_transformation(new_transformation)
```

Ao final de cada treinamento, todos as atividades devem ser encerrados com o método *end()*.

4.2 Análise de sobrecarga da captura de proveniência em PINNs

Para avaliar se a sobrecarga de coleta de dados de proveniência pode impactar negativamente o desempenho das PINNs, fizemos uma comparação do tempo de

execução do treinamento do modelo de configuração 5 do experimento da Seção 5.3 em três cenários, de acordo com a seguinte lista:

1. Execução do treinamento somente com o TensorFlow
2. Execução do treinamento salvando com a biblioteca Pandas ao final da execução.
3. Execução do treinamento salvando com a DNNProv os resultados a cada 100 épocas.

O primeiro cenário representa a execução sem a sobrecarga da coleta de dados de proveniência, que será a base para a comparação do tempo de execução com os outros cenários.

O segundo cenário coleta os dados referentes às métricas de saída do treinamento utilizando a biblioteca Pandas. Para isso foi criado um objeto *DataFrame* com o Pandas que é acrescentado com as métricas de saída do treinamento a cada intervalo de 100 épocas. Ao final do treinamento esse objeto foi persistido em formato csv em conjunto com os hiperparâmetros do modelo. Esse cenário, embora não colete dados de proveniência, representa ações tipicas de cientistas que precisam desses dados para analisar e guardar os dados sobre a execução do treinamento.

O terceiro cenário visa analisar as ações de coleta, estruturação, armazenamento de dados de proveniência e resultados referentes às métricas de avaliação e hiperparâmetros do modelo a cada 100 épocas, utilizando a DNNProv já estendida para os parâmetros de PINNs, que armazena essas informações no SGBD MonetDB.

As execuções foram feitas em um computador pessoal com processador Intel Core i5 8400, 32 GBs de memória ram, GPU Nvidia Gtx 1060 de 6 GBs e sistema operacional Ubuntu 20.04.6. A medição de sobrecarga foi feita nesse ambiente pelo motivo de oferecer mais controle da execução se comparado com ambientes HPC.

Foram feitas quatro execuções ininterruptas do treinamento em 1000, 10000 e 100000 épocas para cada um dos cenários. A primeira execução de cada categoria de épocas foi descartada, pois essa tem o objetivo de inicialização e preparação do sistema, como pré-carregar dados em cache. As Tabelas 4.1,4.2,4.3, mostram os tempos de cada execução para cada um dos cenários, enquanto a Tabela 4.4 mostra a diferença percentual do tempo médio do cenário 2 e 3 comparado com o cenário 1. A Figura 4.2 ilustra visualmente a diferença entre os tempos médios.

Execução	1000 Épocas	10000 Épocas	100000 Épocas
1	0.499	2.406	21.524
2	0.494	2.415	21.930
3	0.495	2.396	21.537

Tabela 4.1: Tempo em minutos para cada execução do cenário 1.

Execução	1000 Épocas	10000 Épocas	100000 Épocas
1	0.508	2.468	22.009
2	0.509	2.448	22.032
3	0.513	2.458	22.102

Tabela 4.2: Tempo em minutos para cada execução do cenário 2.

Execução	1000 Épocas	10000 Épocas	100000 Épocas
1	0.512	2.530	22.898
2	0.517	2.549	22.822
3	0.516	2.545	22.825

Tabela 4.3: Tempo em minutos para cada execução do cenário 3.

Cenário	1000 Épocas	10000 Épocas	100000 Épocas
2	2.88%	2.16%	1.78%
3	3.88%	5.63%	5.48%

Tabela 4.4: Tempo médio da sobrecarga da execução dos cenários 2 e 3 tendo como base de comparação o cenário 1.

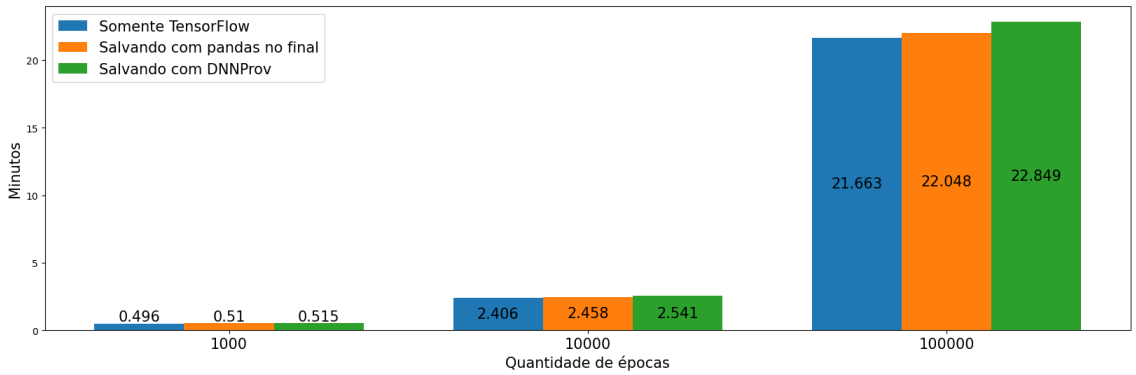

Figura 4.2: Comparação do tempo médio de execução de cada cenário.

Podemos observar que o cenário 2 tem o tempo médio de execução mais rápido comparado com o cenário 3. Isso já era esperado, pois o cenário 2 faz somente um único salvamento ao final do treinamento, enquanto o cenário 3 faz um salvamento a cada 100 épocas. No cenário 3, a DNNProv envia as informações para a DfAnalyzer que assincronamente realiza a persistência no SGBD, usando o processador, sem concorrer com a execução do treinamento. Diferente do cenário 2, o cenário 3 implementa proveniência e também oferece uma vantagem para o cientista de análise

em tempo real do treinamento, e isso se dá ao custo de apenas alguns segundos de processamento.

O intervalo do salvamento das épocas pode ser personalizado utilizando a DNN-Prov. Nesse experimento estamos fazendo o salvamento de uma única observação das métricas de saída a cada intervalo de 100 épocas. Uma possível melhoria para a DNNProv seria salvar ao invés de uma única observação a cada intervalo de época, seria agrupar todas as observações desse intervalo e posteriormente persistir-los no SGBD.

Capítulo 5

Experimentos com dados de proveniência e PINNs

Realizamos três experimentos com objetivos diferentes porém complementares para alcançar o objetivo principal da dissertação. O experimento inicial tem por objetivo avaliar como a proveniência pode ajudar o cientista em um problema de geração e escolha de modelos de uma rede neural convolucional, em um experimento de geração e escolha de modelos no cenário de processamento de imagens. Para isso, utilizamos a Keras-Prov, que é uma extensão direta da biblioteca da DNNProv para captura de dados de proveniência de execuções de DNNs no *framework* Keras. A escolha da Keras-Prov foi feita pois ela utiliza o mesmo *back-end* da DNNProv, compartilhando por exemplo, o mesmo modelo de proveniência representado pela Figura 4.1 e também persiste os dados no SGBD MonetDB. Por ser especializada para execuções no *framework* Keras, sua instrumentação é mais fácil se comparada com a DNNProv. Apesar da Keras-Prov não ser expansível para dados relacionados a PINNs e *framework* de treinamento, esse experimento teve objetivo de ser o ponto de partida para avaliar a captura de dados de proveniência em DNNs.

Já no segundo experimento, avaliamos como a proposta desta dissertação, que aborda a utilização da proveniência para apoiar os cientistas desenvolvedores de PINNs no problema de geração e escolha de modelos, pode ajudar o cientista de PINNs no cenário de escolha do melhor modelo para o problema relacionado à equação eikonal fatorada, utilizando o *framework* do TensorFlow. A fim disso, utilizamos como base a solução de proveniência da DNNProv estendida conforme apresentada na Seção 4.1. Para o terceiro experimento utilizamos a DNNProv em conjunto com o *framework* DeepXDE (LU *et al.*, 2021), que é uma biblioteca que vem ganhando popularidade com os cientistas desenvolvedores de PINNs. Essa é uma biblioteca que não possui implementação de proveniência, que foi recentemente desenvolvida e está sendo frequentemente atualizada com o objetivo de facilitar a criação de PINNs. Esse experimento avalia a execução da extensão da DNNProv em diferentes

frameworks. No experimento da DeepXDE trabalhamos com um exemplo simples fornecido pela biblioteca, o problema inverso para a equação de Poisson com campo de força desconhecido.

Este capítulo está dividido em 4 seções. A Seção 1 descreve o ambiente computacional de alto desempenho utilizado para executar os experimentos. A Seção 2 apresenta o primeiro experimento, que avalia como a utilização de proveniência ajuda no desenvolvimento de uma DNN de classificação de imagem utilizando o *framework* Keras. A Seção 3 mostra como a proveniência apoia o cientista de PINNs na tarefa de análise e escolha de configurações de modelos envolvendo a equação eikonal utilizando o *framework* TensorFlow. Por fim, a Seção 4 analisa a extensibilidade da solução de proveniência para uma PINN que aborda um problema de equação de Poisson no *framework* DeepXDE.

5.1 Ambiente computacional

Para os três experimentos utilizamos o super computador Sdumont[1], que possui uma capacidade de processamento instalada de aproximadamente 5,1 Petaflop/s ($5.1*10^{15}$ operações de ponto flutuante por segundo), utilizando uma configuração híbrida de nós computacionais que incorpora diferentes arquiteturas de processamento paralelo. Os 1134 nós do SDumont estão conectados por uma rede InfiniBand, que oferece alta taxa de transferência e baixa latência para a comunicação entre processos e acesso ao sistema de arquivos. Além disso, o SDumont possui um sistema de arquivos paralelo Lustre, com uma capacidade de armazenamento bruta de cerca de 1,7 PBytes, e também tem disponível um sistema de arquivos secundário com capacidade bruta de 640 TBytes.

Utilizamos para os nossos experimentos com opção de exclusividade habilitada, o nó com nome de GDL, que é especializado para tarefas de inteligência artificial. Esse nó conta com 2x Intel Xeon Skylake Gold 6148 (20c @2,4GHz), 8x NVIDIA Tesla V100-16GB com NVLink e 384Gb de memória RAM. Apesar de oferecer 8 GPUs, nossos experimentos inicialmente usaram apenas uma das GPUs.

[1] https://sdumont.lncc.br/

5.2 Análise de consultas com dados de proveniência na Keras-Prov

5.2.1 A ferramenta Keras-Prov

A Keras-Prov (KUNSTMANN *et al.*, 2021; PINA *et al.*, 2021) é uma ferramenta com o objetivo de realizar a captura dos hiperparâmetros e métricas de avaliação de DNNs durante o treinamento utilizando proveniência, seguindo o modelo representado pela Figura 4.1. Ela usa por base a DNNProv, visando a facilitar a interação do usuário com o serviço de coleta de proveniência. A vantagem em relação à DNNProv é que o usuário não precisa instrumentar a proveniência no *script* de treinamento da DNN. A desvantagem é que os hiperparâmetros e métricas de avaliação a serem monitorados junto à proveniência são pré-programados e sua implementação necessita de uma alteração no código do Keras.

Como a Keras-Prov utiliza o mesmo *back-end* da DNNProv, suas dependências de *softwares* são semelhantes. Essas dependências são especificados no repositório das autoras[2] e são:

- Java - Linguagem utilizada para o funcionamento da DfAnalyzer
- MonetDB - SGBD utilizado para persistir os dados
- DfAnalyzer (SILVA *et al.*, 2018) - RESTful api para geração de dados de proveniência
- dfa-lib-python (CAMPOS, 2018) - Integração da DfAnalyzer para a linguagem Python
- TensorFlow 2.2.0 - Biblioteca para tarefas de aprendizado de máquina
- Keras modificado - API de alto nível do TensorFlow para DNNs modificada com classes de proveniência disponível no repositório das autoras

Após essas instalações, para utilizar a Keras-Prov primeiramente devemos executar o sistema da DfAnalyzer, que será usada pela biblioteca dfa-lib-python para fazer comunicação entre o *script* de execução do treinamento com o SGBD MonetDB e persistir os dados utilizando o modelo de dados de proveniência definido conforme mostra a Figura 4.1.

A Keras-Prov foi feita com o objetivo de a realizar a instrumentação dos dados a serem capturados com proveniência de maneira automática, fazendo isso com os dados mais comuns de DNN. Para isso, basta que durante a criação do modelo o

[2]https://github.com/dbpina/keras-prov

cientista preencha um dicionário informando quais informações deverão ser persistidas. Para isso, basta passar os hiperparâmetros como chave e informar como valor *True* caso o usuário queira que esse valor seja persistido, ou *False* caso contrário. Após a criação desse dicionário, é necessário informá-lo para o método do modelo, em conjunto com o identificador do experimento *dataflow_tag*, usando o método *provenance*, como é ilustrado nas linhas 2-9 a seguir.

```
1  model = Sequential(. . .)
2  hyps = {"OPTIMIZER_NAME": True,
3  "LEARNING_RATE": True,
4  "DECAY": False,
5  "MOMENTUM": False,
6  "NUM_EPOCHS": True,
7  "BATCH_SIZE": True,
8  "NUM_LAYERS": True}
9  model.provenance(dataflow_tag = "prov-alexnet-df", hyps = hyps)
10 model.compile(. . .)
11 model.fit(. . .)
```

Neste experimento iremos capturar hiperparâmetros referentes ao nome do otimizador, taxa de aprendizado, número de épocas, tamanho do *batch* e número de camadas. A Keras-Prov por padrão irá coletar os dados de saída a cada época, como a métrica acurácia, que mede a proporção de predições corretas em relação ao total de predições; e o valor da função de perda, que tem objetivo de ser minimizada ao decorrer do treinamento da DNN. Como na DNNProv, a Keras-Prov persiste os hiperparâmetros de entrada em uma entidade e as métricas de avaliação de saída em outra. Existe identificador que as une, isso é, para cada saída do treinamento é fácil rastrear quais foram os dados de entrada que geraram esse resultado. A Keras-Prov também faz essa captura de maneira online durante o treinamento, o usuário não precisa esperar o treinamento terminar para começar a fazer a análise dos resultados, basta acessar o MonetDB e realizar suas consultas via SQL. A estrutura de código apresentada anteriormente foi utilizada para gerar os sete modelos desse experimento, modificando apenas o otimizador para cada treinamento.

5.2.2 Arquitetura

A AlexNet é uma arquitetura de rede neural convolucional desenvolvida em 2012 por KRIZHEVSKY *et al.* (2012). Ela foi criada para a tarefa de classificação de imagens no conjunto de dados ImageNet (DENG *et al.*, 2009), que consiste em milhões de imagens organizadas em diferentes categorias. Essa arquitetura obteve uma vitória notável na competição ImageNet Large Scale Visual Recognition Challenge (ILSVRC) de 2012. A AlexNet possui uma estrutura com oito camadas, sendo

as primeiras cinco camadas convolucionais seguidas por três camadas totalmente conectadas, como é ilustrado na Figura 5.1.

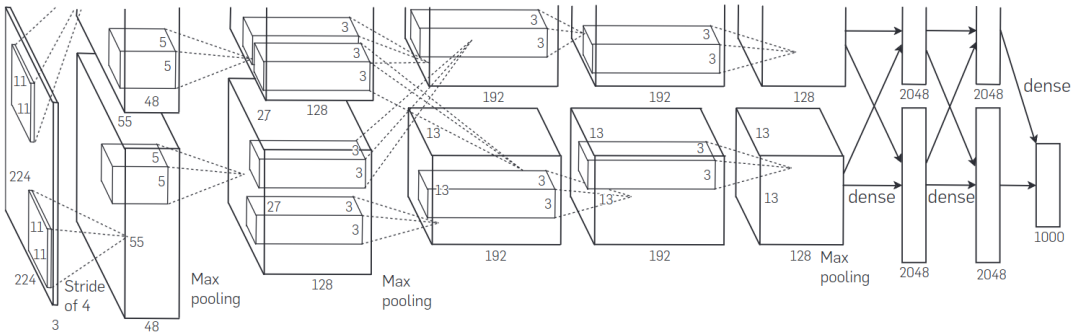

Figura 5.1: Arquitetura da Alexnet (KRIZHEVSKY *et al.*, 2012).

Essa arquitetura foi inovadora para época, sendo uma das responsáveis por impulsionar o avanço da aprendizagem profunda ao obter resultados até então não alcançados por DNNs na ILSVRC, superando os métodos tradicionais de aprendizado de máquina. Seu sucesso ajudou a abrir caminhos para o desenvolvimento de DNNs ainda mais profundas e complexas no campo da visão computacional.

5.2.3 Experimento

O experimento é um exemplo de aplicação fornecido pelo repositório da Keras-Prov, tem como base de dados o Oxford Flowers 17 (NILSBACK e ZISSERMAN, 2006), que é comumente utilizado para *benchmark* de modelos de classificação de imagens. Essa base de dados possui 1360 imagens coloridas, contando com 17 espécies de flores diferentes, onde cada uma possui 80 imagens com diferentes variações de iluminação, ângulos de câmera e escalas, como é ilustrado na Figura 5.2. Cada imagem é acompanhada de um único rótulo que indica a categoria de flor que representa.

Nesse presente experimento temos como foco avaliar como a biblioteca da Keras-Prov pode ajudar cientistas na geração e escolhas de modelos de classificação de imagens utilizando o *framework* Keras. Para isso, temos como aplicação a escolha de modelos para arquitetura da Alexnet. Nesse caso, o cientista busca encontrar o melhor modelo com base nos hiperparâmetros apresentados na Tabela 5.1, que conta com a variação de otimizadores.

Após as execuções dos treinamentos feitas utilizando a Keras-Prov, podemos verificar os identificadores dos experimentos através das *tags* presentes na tabela *dataflow* no MonetDB, como mostra a Figura 5.3. Como para todos as execuções dos treinamentos utilizamos o mesmo identificador *prov-alexnet-df*, teremos apenas esse exibido.

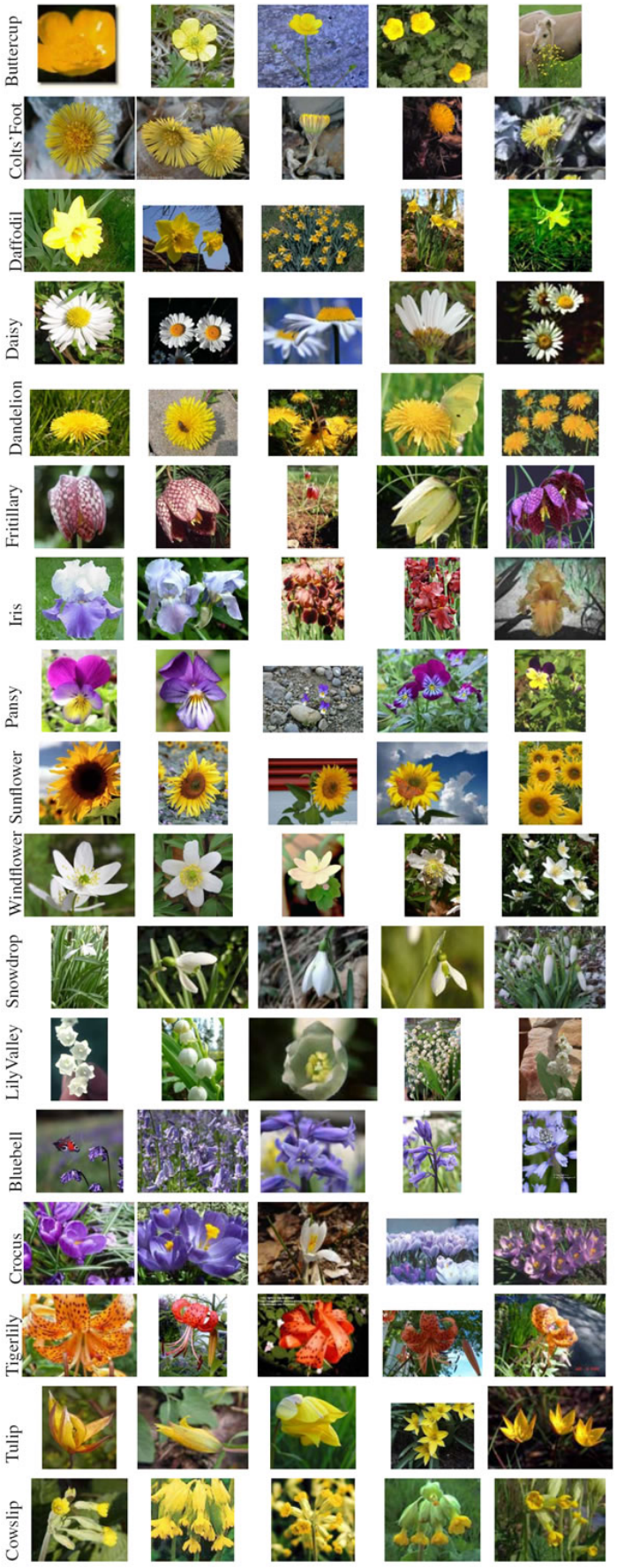

Figura 5.2: Exemplo de 5 imagens para cada uma das 17 categorias (NILSBACK e ZISSERMAN, 2006).

Tabela 5.1: Conjunto de hiperparâmetros para serem avaliados.

Épocas	Taxa de aprendizado	Ativação	Dropout	Otimizadores
500	0.0001	Relu	0.4	SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam

```
sql>select * from dataflow;
+-------+------------------+
| id    | tag              |
+=======+==================+
|     1 | prov-alexnet-df |
+-------+------------------+
```

Figura 5.3: Identificador relativo ao experimento da Keras-Prov.

Através dos identificadores, podemos ter acesso as entidades geradas pelos treinamentos. No caso do presente experimento, foi gerada a entidade *itrainingmodel* que armazena as informações de entrada ligadas aos hiperparâmetros, que é ilustrada pela Figura 5.4. E podemos encontrar os dados de saída do treinamento através da entidade *otrainingmodel*, que é ilustrada pela Figura 5.5.

```
sql>select * from "prov-alexnet-df".itrainingmodel;
+------+-------+--------------------+-----------+--------------------------+--------------------------+--------------------------+
| id   | train | itrain_timestamp   | optimizer | learning_rate            | num_epochs               | batch_size               |
:      : ingmo :                    : _name     :                          :                          :                          :
:      : del_t :                    :           :                          :                          :                          :
:      : ask_i :                    :           :                          :                          :                          :
:      : d     :                    :           :                          :                          :                          :
+======+=======+====================+===========+==========================+==========================+==========================+
|    1 |     1 | 1687654188.5094428 | Adam      |                   0.0001 |                      500 |                       64 |
|    2 |     2 | 1687654841.5465238 | SGD       |                   0.0001 |                      500 |                       64 |
|    3 |     3 | 1687655402.6074839 | RMSprop   |                   0.0001 |                      500 |                       64 |
|    4 |     4 | 1687656028.1500416 | Adadelta  |                   0.0001 |                      500 |                       64 |
|    5 |     5 | 1687656695.3888516 | Adagrad   |                   0.0001 |                      500 |                       64 |
|    6 |     6 | 1687657267.0047503 | Adamax    |                   0.0001 |                      500 |                       64 |
|    7 |     7 | 1687657893.526185  | Nadam     |                   0.0001 |                      500 |                       64 |
+------+-------+--------------------+-----------+--------------------------+--------------------------+--------------------------+
```

Figura 5.4: Entidade *itrainingmodel* relativa ao experimento da Alexnet.

```
sql>select * from "prov-alexnet-df".otrainingmodel limit 5;
+------+-------+--------+-------+--------------------+--------------------+----------+--------------------+--------------------+--------------------+--------------------+
| id   | train | adapta | testin | otrain_timestamp   | elapsed_time       | epoch_id | loss               | accuracy           | val_loss           | val_accuracy       |
:      : ingmo : tion_t : gmodel :                    :                    :          :                    :                    :                    :                    :
:      : del_t : ask_id : _task_ :                    :                    :          :                    :                    :                    :                    :
:      : ask_i :        : id     :                    :                    :          :                    :                    :                    :                    :
:      : d     :        :        :                    :                    :          :                    :                    :                    :                    :
+======+=======+========+=======+====================+====================+==========+====================+====================+====================+====================+
|    1 |     1 |   null |  null | 1687654201.2556777 | 12.66431212425232  |        0 |  2.9689305109136246 |         0.16360295 |  2.8265088586246265 |  0.11397058516740799 |
|    2 |     1 |   null |  null | 1687654202.4935285 | 1.2208903898162842 |        1 |   2.164317628916572 |          0.3290441 |  2.8342331016764923 |  0.11029411852359772 |
|    3 |     1 |   null |  null | 1687654203.7225525 | 1.2215189933776855 |        2 |  1.7544786228853113 |         0.40808824 |  2.8682018448324764 |  0.07352941483259201 |
|    4 |     1 |   null |  null | 1687654204.959437  | 1.2301015853881836 |        3 |  1.4564267467049992 |         0.53768384 |  2.937327356894858  |               0.0625 |
|    5 |     1 |   null |  null | 1687654206.1942954 | 1.2278773784637451 |        4 |  1.2132463034461527 |          0.5965074 |  3.0516934394836426 |  0.04779411852359772 |
+------+-------+--------+-------+--------------------+--------------------+----------+--------------------+--------------------+--------------------+--------------------+
```

Figura 5.5: Extrato de 5 linhas da entidade *otrainingmodel* relativa ao experimento da Alexnet.

Observe que a entidade *itrainingmodel* possui um identificador para cada execução das configurações treinadas. Esse identificador é presente na entidade *otrainingmodel*, assim sendo fácil associar cada configuração com seus resultados. No caso

da entidade *otrainingmodel*, além do identificador de treinamento, também é possível verificar que existem identificadores para adaptação e para teste. No presente experimentos esses se encontram *null* pois não foram executados procedimentos de adaptação e teste no *script* de treinamento das DNNs.

Para fazer a análise comparativa dos sete modelos treinados, iremos realizar consultas (C) na nossa base de dados de proveniência com base em consultas populares (SCHELTER *et al.*, 2017) MIAO *et al.* (2017) TSAY *et al.* (2018) GHARIBI *et al.* (2019).

C$_1$ Quais foram os melhores valores da métrica acurácia para cada otimizador no treinamento e na validação?

C$_2$ Quais foram os melhores valores da função de perda para cada otimizador no treinamento e na validação?

C$_3$ Qual otimizador convergiu mais rápido para o seu melhor resultado de acurácia na validação?

C$_4$ Quais foram as médias de tempo do treinamento das épocas para cada otimizador?

C$_5$ Como foram os comportamentos da acurácia e da função de perda ao longo das épocas?

Tais consultas podem ser realizadas utilizando a linguagem SQL diretamente no MonetDB usando as entidades *itrainingmodel* e *otrainingmodel*. Também podemos realiaza-las no Python através da biblioteca *pymonetdb*, que faz a conexão e comunicação com o SGBD. Dessa forma, podemos ter como resultado das consultas objetos Python e assim utilizar todos os seus benefícios oferecidos para visualização.

C$_1$ Quais foram os melhores valores da métrica acurácia para cada otimizador no treinamento e na validação?

Observando a Tabela 5.2, a maioria dos otimizadores obteve uma taxa de acerto de 100% nos dados de treinamento. No entanto, o Adadelta se destacou negativamente, apresentando um desempenho muito inferior comparado aos demais. Ao levamos em conta os dados de validação, nenhum dos otimizadores alcançou uma acurácia superior a pelo menos 75%, o que sugere a ocorrência de sobreajuste aos dados de treinamento. Vale ressaltar que o otimizador Adadelta mostrou-se ineficiente também nos dados de validação, indicando que não é uma escolha adequada para este experimento.

Tabela 5.2: Resultados da Consulta 1.

Otimizador	Acurácia Treino	Acurácia Validação
Adadelta	0.34	0.37
Adagrad	1.0	0.64
Adam	1.0	0.7
Adamax	1.0	0.71
Nadam	1.0	0.73
RMSprop	1.0	0.68
SGD	0.99	0.63

C_2 Quais foram os melhores valores da função de perda para cada otimizador no treinamento e na validação?

Ao analisarmos os valores da função de perda apresentados na Tabela 5.3, podemos notar que os otimizadores Adadelta e SGD apresentaram os piores resultados em relação aos dados de treinamento. No entanto comparando com os dados de validação, todos os otimizadores performaram de maneira mais parecida. Apesar do SGD ter sido o pior com os dados de treino, foi o melhor com os dados de validação.

Tabela 5.3: Resultados da Consulta 2.

Otimizador	Função de perda Treino	Função de perda Validação
Adadelta	$2.13 * 10^{0}$	$2.04 * 10^{0}$
Adagrad	$8.79 * 10^{-3}$	$1.30 * 10^{0}$
Adam	$2.49 * 10^{-5}$	$1.43 * 10^{0}$
Adamax	$1.04 * 10^{-5}$	$1.29 * 10^{0}$
Nadam	$2.03 * 10^{-5}$	$1.31 * 10^{0}$
RMSprop	$2.44 * 10^{-5}$	$1.72 * 10^{0}$
SGD	$0.11 * 10^{0}$	$1.24 * 10^{0}$

C_3 Qual otimizador convergiu mais rápido para o seu melhor resultado de acurácia na validação?

Levando em consideração que as configurações foram executadas por 500 épocas, podemos observar pela Tabela 5.4 que os otimizadores Adagrad, Adam, Nadam, RMSProp alcançaram seus melhores valores de acurácia na validação relativamente cedo. Em contrapartida, os otimizadores AdaDelta, Adamax e SGD tiveram seu ápice mais próximo ao fim do treinamento. Isso pode indicar que ao aumentar a quantidade de épocas, esses otimizadores têm potencial de obter melhores valores de acurácia para os dados fora do conjunto de treinamento.

Tabela 5.4: Resultados da Consulta 3.

Otimizador	Acurácia validação	Época de obtenção	Tempo gasto
Adadelta	0.37	495	647.78
Adagrad	0.64	121	138.63
Adam	0.7	292	376.14
Adamax	0.71	441	540.28
Nadam	0.73	116	149.27
RMSprop	0.68	152	188.13
SGD	0.63	494	541.17

C_4 Quais foram as médias de tempo do treinamento das épocas para cada otimizador?

Em relação ao tempo médio de execução das épocas, que é ilustrado pela Figura 5.6, podemos ver que o Adadelta e Adam foram os otimizadores que mais necessitaram de tempo de processamento. Já o Nadam, Adamax e RMSProp tiveram performances parecidas entre si, e o SGD seguido do Adagrad foram os mais rápidos.

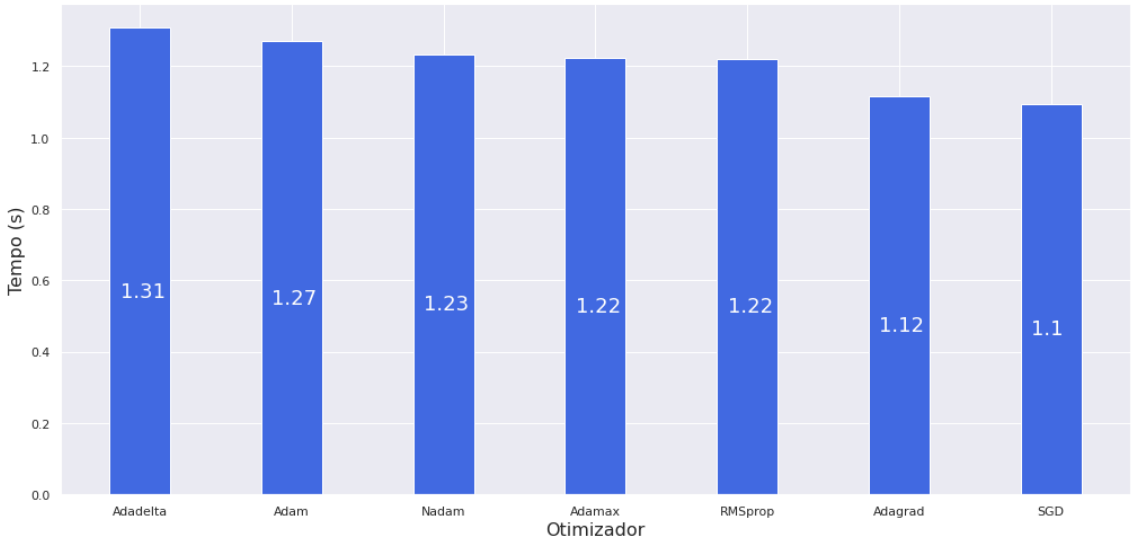

Figura 5.6: Média do tempo das épocas por otimizador.

C_5 Como foram os comportamentos da acurácia e da função de perda ao longo das épocas?

Observando a Figura 5.7, os otimizadores Adamax, RMSprop, Adam, NAdam, Adagrad obtiveram comportamento parecido tanto para a acurácia e para função de perda no conjunto de dados do treinamento, parecendo se estabilizar perto da época de número 150. É possível também observar grandes variações nos valores dessas

métricas perto das épocas 110 e 440 para o Adam; 200 e 450 para o Nadam. Já os otimizadores SGD e Adadelta tiveram comportamentos bem diferente dos outros. O Adadelta não conseguiu convergir para valores aceitáveis dessas métricas durante as 500 épocas, já o SGD, conseguiu obter melhores resultados e acompanhar os demais perto do fim.

Em relação aos dados da validação, houve uma maior discrepância no comportamento dos sete otimizadores, ficando difícil estabelecer um ponto em que pelo menos alguns deles convergem entre si. Para a acurácia, o SGD conseguiu também se aproximar as estimativas dos demais, porém o Adadelta continuo com um baixo desempenho. Para a função de perda, todos os otimizadores se comportaram de maneira mais parecida. Vale observar também que as grande variações se repetiram para o Adam e NAdam nos mesmos intervalos de épocas.

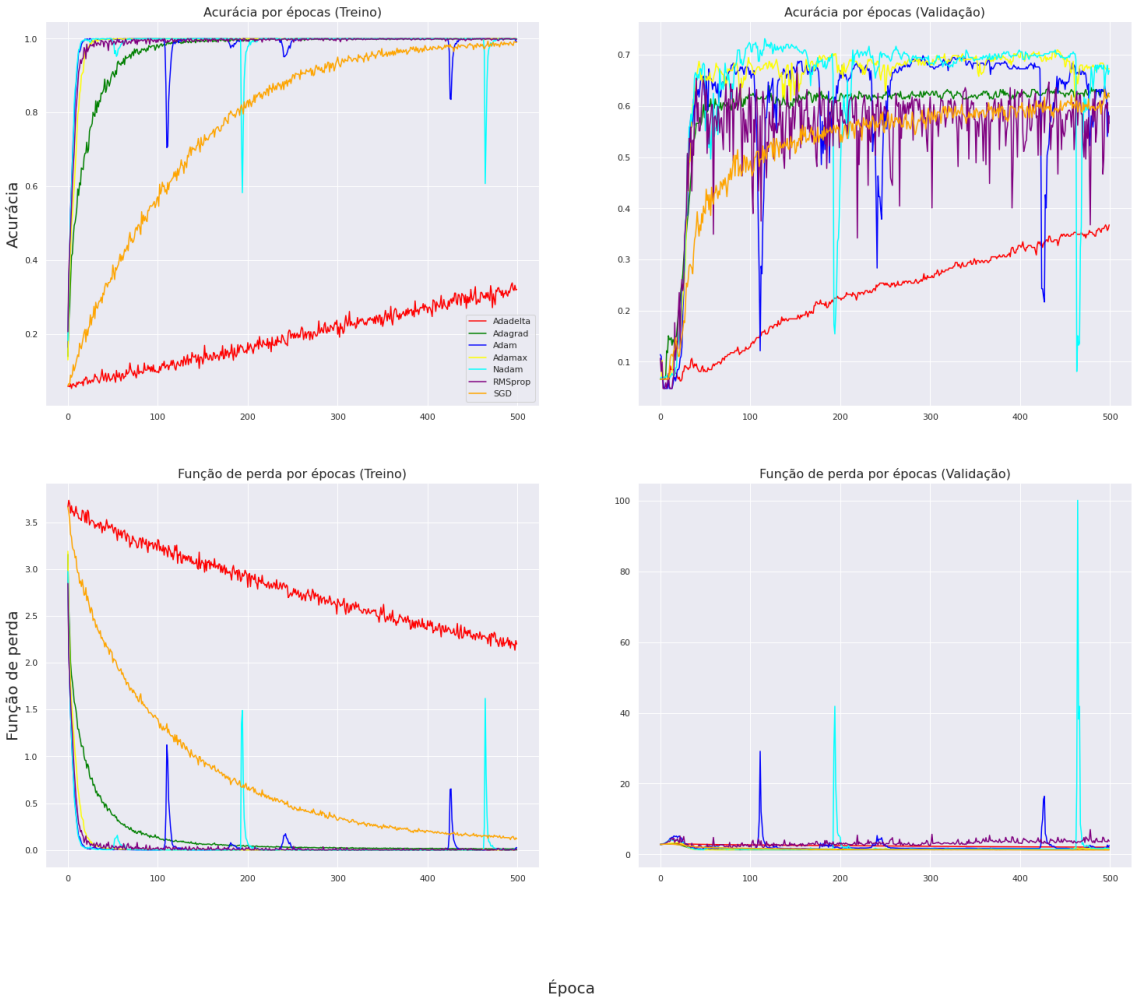

Figura 5.7: Acurácia e função de perda por épocas em treino e validação.

Pela Figura 5.8, fica mais claro observar a variação da acurácia dos otimizadores com os dados de validação. Além de possuir os menores valores dessa métrica, o Nadam possui também a maior variação entre os otimizadores, se destacando

negativamente. Também é possível notar uma grande quantidade de *outliers* para os demais otimizadores. O Nadam foi o otimizador que mais se aproximou de ter uma mediana de 70%, que comparado com os outros, se destaca positivamente.

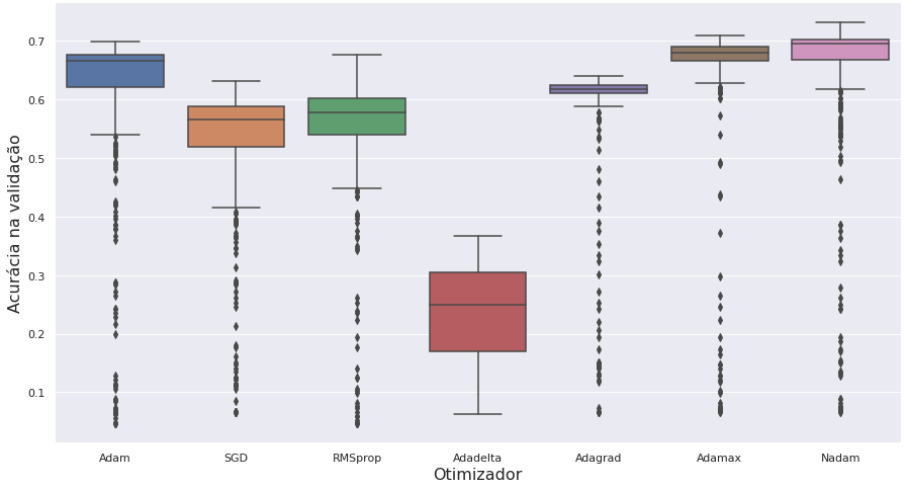

Figura 5.8: Variação da acurácia na validação.

Nesse experimento, o cientista pôde comparar sete diferentes otimizadores para o problema de escolha de modelos realizando consultas na base de dados de proveniência. O cientista pode ter a conclusão que o otimizador Nadam é promissor comparados com os demais, pois obteve o melhor valor de acurácia para os dados de validação. Porém mesmo sendo o melhor, ainda apresenta problemas com sobreajuste. Para os próximos passos, o cientista pode fixar como otimizador o Nadam e variar outros hiperparâmetros da DNN, realizando ajustes finos com objetivo de ter melhores métricas e minimizar o sobreajuste. Para apoiar nessa tarefa, o cientista pode continuar utilizando a Keras-Prov para a persistência dos dados do modelo. Realizando esse processo de melhorias em etapas, com o cientista no controle das decisões, o processo de escolha interativa de modelo inserindo o humano no *loop* está sendo aplicado.

5.3 Estudo de caso: Proveniência com a PINN eikonal no TensorFlow

5.3.1 Definição do problema

Para esse experimento, iremos ter como base o experimento realizado por SILVA *et al.* (2021), onde os autores desenvolveram uma PINN para resolver a Equação Eikonal Fatorada (EEF), que descreve fenômenos como propagação de ondas para meios acústicos e elásticos, bem como meios eletromagnéticos (DEBNATH, 2012). Nosso objetivo é realizar a geração de mais modelos, com novos conjuntos de hiperparâmetros, seguindo a ideia da escolha interativa de modelo inserindo o humano no *loop* para servir de apoio para a escolha do melhor modelo gerado com apoio de proveniência. Esta PINN retrata um tipo de problema de tomografia sísmica. Nesse experimento, além de estimar o tempo de transito relacionado ao problema direto, uma rede auxiliar é também treinada para resolver o problema inverso associado a velocidade de propagação de onda, conforme é mostrado na Figura 5.9.

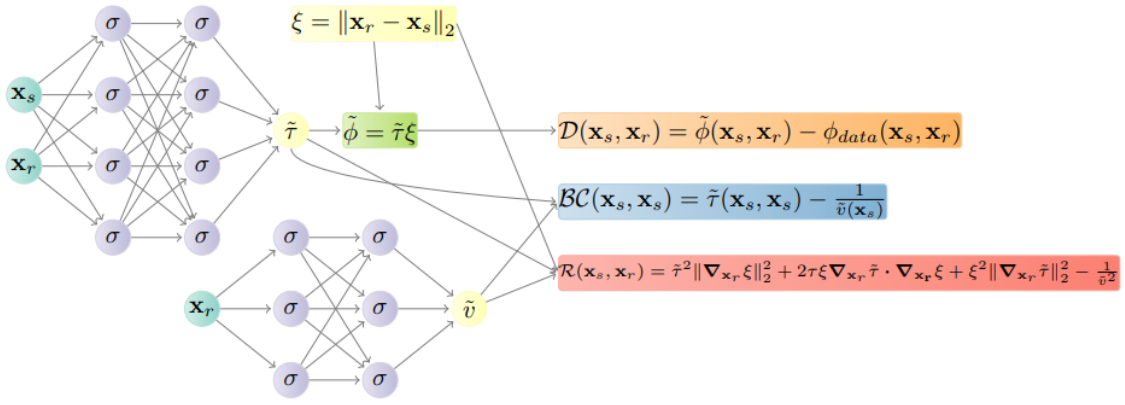

Figura 5.9: Esquema de PINN para resolver o problema inverso da EEF. $\phi(\mathbf{x}_s, \mathbf{x}_r)$ representa os tempos de transito, e $\upsilon(\mathbf{x}_r)$, a velocidade de propagação da onda no meio acústico, são aproximados por duas redes neurais diferentes, suas aproximações são denotadas por $\tilde{\phi}(\mathbf{x}_s, \mathbf{x}_r)$ e $\tilde{\upsilon}(\mathbf{x}_r)$ respectivamente. Essas aproximações são utilizadas pelos componentes relacionados à função de perda (SILVA *et al.*, 2021).

As PINNs podem ser definidas e treinadas usando os ambientes clássicos de DNNs, como TensorFlow e PyTorch. Na prática, o cientista especialista em PINNs e na Física em questão precisa configurar, por ex., o TensorBoard para exibir os valores dos componentes da função de perda por meio de rotinas de sumarização de dados do TensorFlow. Essa estratégia requer a programação de um *script* específico para relacionar informações em diferentes arquivos e persisti-los em arquivos tipo CSV. Geralmente os resultados para serem analisados só estarão disponíveis ao final da execução. Além disso, em problemas de análise de múltiplos modelos é

necessário planejar a organização desses arquivos/diretórios de diversos resultados das configurações de modelos, o que dificulta a análise comparativa e a escolha do melhor entre eles. Para esse experimento, iremos utilizar a solução de proveniência fornecida pela biblioteca DNNProv com a nossa extensão em conjunto com o *framework* TensorFlow. A implementação da DNNProv para esse experimento foi feita de acordo com o apresentado na Seção 4.1.

5.3.2 Experimento

O conjunto de dados de entrada utilizado para aplicação das PINNs foi o mesmo de SILVA *et al.* (2021), sendo referente a dados sísmicos e de radar de penetração no solo. A métrica de qualidade de ajuste é a R^2, que é conhecida como coeficiente de determinação. Este experimento explorou configurações em escala maior. A ilustração da solução real é apresentada na Figura 5.10. Conforme a Figura 5.9, foram utilizadas duas redes neurais, uma para estimar o tempo de transito (TT), relacionado ao problema direto, e outra para estimar a velocidade de propagação da onda (Vel), relacionado ao problema inverso. Essas redes estão indiretamente conectadas para o cálculo da função de perda. Uma versão deste experimento foi publicado pelo autor desta dissertação no Simpósio Brasileiro de Banco de Dados de 2022 (DE OLIVEIRA *et al.*, 2022).

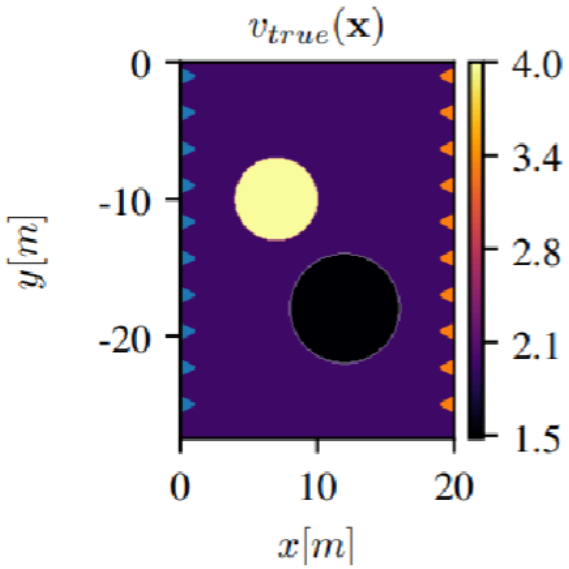

Figura 5.10: Representação gráfica da solução real. Modelo de velocidade real terrestre corresponde para um modelo de velocidade de fundo com $v_{true}(\mathbf{x})$ = 2,0 [km / s] com duas inclusões com $v_{true}(\mathbf{x})$ = 4,0 [km / s] (canto superior esquerdo) e $v_{true}(\mathbf{x})$ = 1,5 [km / s] (canto inferior direito) (SILVA *et al.*, 2021).

Os pesos w da função de perda apresentados na Equação 2.2 foram definidos de acordo com a Equação 5.1.

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{T}) = 1 \cdot \mathcal{L}_{\mathcal{R}}(\boldsymbol{\theta};\mathcal{T}_{\mathcal{R}}) + 1 \cdot \mathcal{L}_{\mathcal{BC}}(\boldsymbol{\theta};\mathcal{T}_{\mathcal{BC}}) + 25 \cdot \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta};\mathcal{T}_{\mathcal{D}}) \tag{5.1}$$

Nesse experimento, para a atribuição desses pesos foi usada a intuição do cientista, baseada em experimentos passados. Foram utilizados os hiperparâmetros *batch size* = 3000, número de épocas = 400000 e taxa de aprendizado com a função de decaimento exponencial, com parâmetros definidos na Tabela 5.5, esta atribui um valor inicial para a taxa e a diminui ao decorrer das épocas.

Função	Decaimento	Passos	Aprendizado Inicial	Aprendizado Final
Dec. Exponencial	0.99	1000	0.001	0.000018

Tabela 5.5: Parâmetros do decaimento exponencial da taxa de aprendizado.

A biblioteca de proveniência DNNProv foi adotada para auxiliar na coleta de dados com proveniência no SGBD, persistindo as informações de saída do treinamento a cada intervalo de 100 épocas. O cientista utiliza o processo de escolha interativa de modelo com humano no *loop* para buscar a melhor configuração de hiperparâmetros. O experimento foi dividido em duas rodadas, ao final de cada rodada o cientista busca fazer alterações para obter melhores desempenhos para a métrica de avaliação do modelo R^2 e função de perda. Para cada rodada, o cientista faz consultas à base de dados de proveniência para avaliar o desempenho de cada configuração, essas consultas são:

1. Quais são as configurações das PINNs treinadas?
2. Quais são os maiores valores de R^2, qual é a época e tempo gasto para obtenção?
3. Quais são os menores valores da função perda e seus componentes, qual é a época e tempo gasto para obtenção?

Primeira Rodada

Para a primeira rodada foram testadas diferentes funções de ativação e dois otimizadores, Adam e RMSprop, para uma arquitetura específica.

Tabela 5.6: Quais são as configurações das PINNs treinadas para a primeira rodada?

ID	Otimizador	Ativação TT	Ativação Vel	N.º de Neurônios TT	N.º de Neurônios Vel	N.º de Camadas intermediárias TT	N.º de Camadas intermediárias Vel
1	Adam	tanh	tanh	20	32	8	4
2	Adam	relu	relu	20	32	8	4
3	Adam	sigmoid	sigmoid	20	32	8	4
4	Adam	relu	tanh	20	32	8	4
5	RMSProp	tanh	tanh	20	32	8	4

Tabela 5.7: Quais são os maiores valores de R^2, qual é a época e tempo gasto para obtenção na primeira rodada?

ID	Máximo R^2	Época de obtenção	Tempo até obtenção(m)	Média 100 épocas(s)
5	0.470	398600	100.912	1.519
1	0.377	88000	22.293	1.520
3	0.105	397200	100.756	1.522
2	0.093	86300	20.165	1.402
4	0.077	3400	0.790	1.395

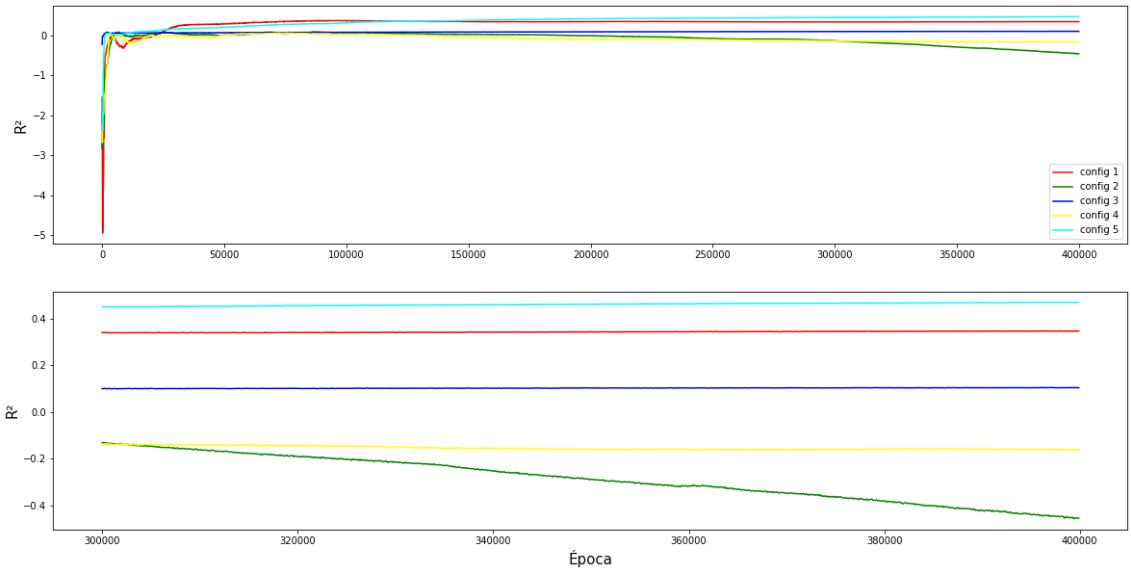

Figura 5.11: Valor da métrica R^2 para todas as épocas (Superior); Foco nas últimas 100.000 épocas (Inferior). Referente à primeira rodada.

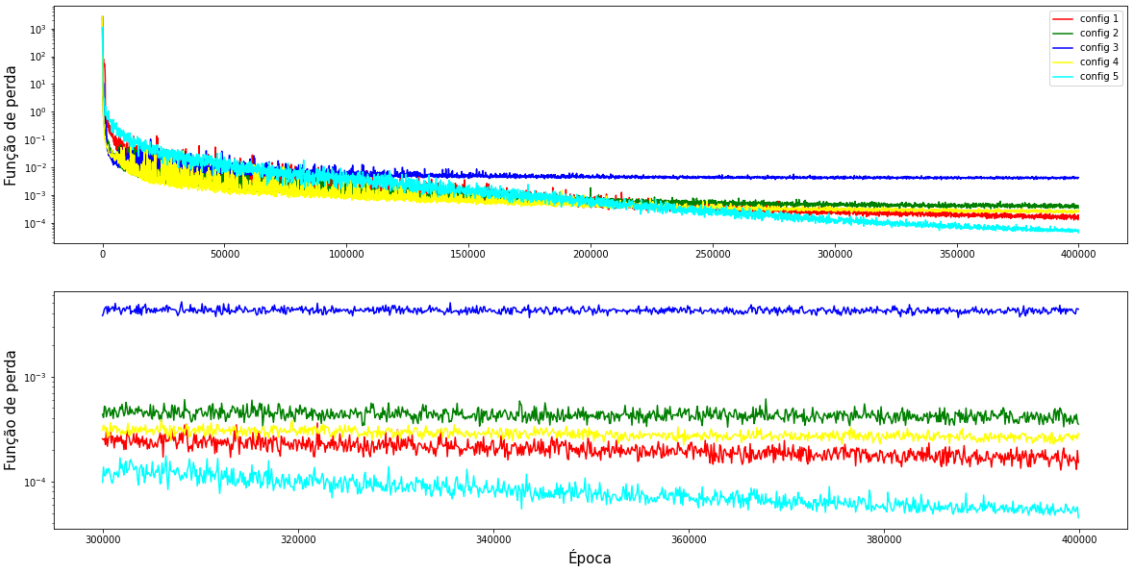

Figura 5.12: Valor da função de perda para todas as épocas (Superior); Foco nas últimas 100.000 épocas (Inferior). Referente à primeira rodada.

Tabela 5.8: Quais são os menores valores da função perda e seus componentes, qual é a época e tempo gasto para obtenção para primeira rodada?

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Época de obtenção	Tempo até obtenção(m)
5	$4.503 \cdot 10^{-5}$	3.72710^{-5}	$4.066 \cdot 10^{-10}$	$3.102 \cdot 10^{-7}$	399900	101.241
1	$1.283 \cdot 10^{-4}$	3.69210^{-5}	$3.771 \cdot 10^{-8}$	$3.654 \cdot 10^{-6}$	398100	100.852
4	$2.229 \cdot 10^{-4}$	1.25210^{-4}	$7.298 \cdot 10^{-9}$	$3.910 \cdot 10^{-6}$	359500	83.584
2	$3.290 \cdot 10^{-4}$	1.67710^{-4}	$1.527 \cdot 10^{-9}$	$6.449 \cdot 10^{-6}$	364000	85.055
3	$3.636 \cdot 10^{-3}$	5.86110^{-4}	$2.212 \cdot 10^{-8}$	$1.220 \cdot 10^{-4}$	343700	87.185

Pelos resultados das consultas fornecidas pelas Tabelas 5.6 5.7 5.8, e pela ilustração das Figuras 5.11 5.12 o cientista pode tirar a conclusão que as configurações 1 e 5 se destacaram. Comparando essas duas, além da configuração 5 alcançar um valor maior de R^2, atingiu seu máximo ao fim do treinamento, enquanto a configuração 1 atingiu seu máximo relativamente cedo e não melhorou sua performance desde então.

Segunda Rodada

Para essa nova rodada, o cientista mantém os hiperparâmetros relacionados a configuração 1 e 5, porém agora o cientista faz mudanças relacionadas a topologia das redes TT e Vel, modificando a quantidade de camadas intermediárias e neurônios presentes em cada uma delas.

Tabela 5.9: Quais são as configurações das PINNs treinadas para a segunda rodada?

ID	Otimizador	Ativação TT	Ativação Vel	N.º de Neurônios TT	N.º de Neurônios Vel	N.º de Camadas intermediárias TT	N.º de Camadas intermediárias Vel
6	Adam	tanh	tanh	40	64	16	8
7	RMSProp	tanh	tanh	40	64	16	8
8	Adam	tanh	tanh	20	32	16	8
9	RMSProp	tanh	tanh	20	32	16	8

Tabela 5.10: Quais são os maiores valores de R^2, qual é a época e tempo gasto para obtenção na segunda rodada?

ID	Máximo R^2	Época de obtenção	Tempo até obtenção(m)	Média 100 épocas(s)
8	0.380	399800	174.046	2.612
6	0.219	8200	3.664	2.681
7	0.204	37400	17.011	2.729
9	0.194	124100	54.914	2.655

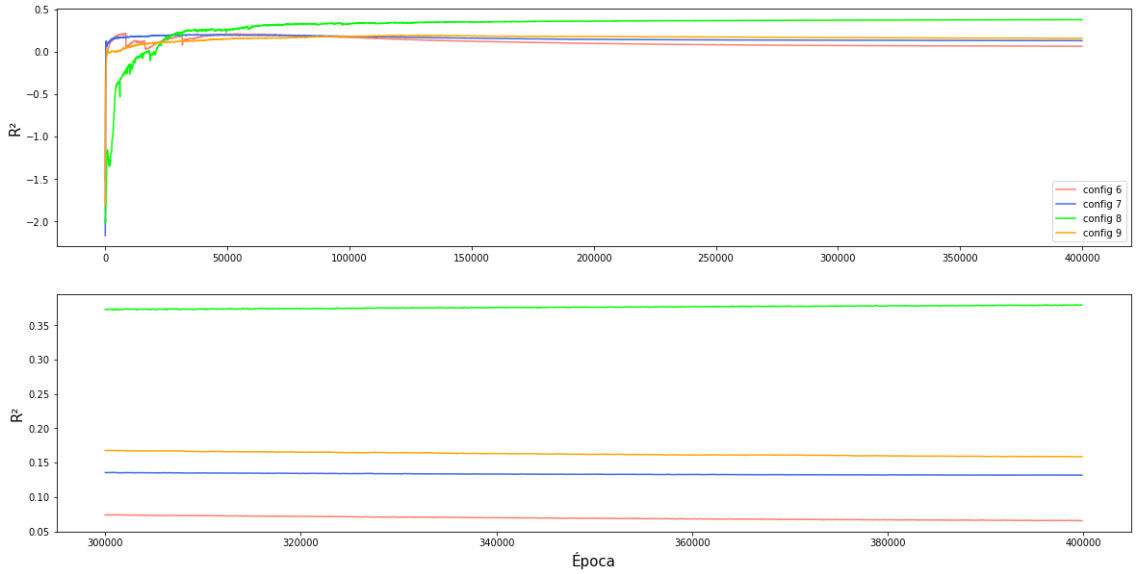

Figura 5.13: Valor da métrica R^2 para todas as épocas (Superior); Foco nas últimas 100.000 épocas (Inferior). Referente à segunda rodada.

Tabela 5.11: Quais são os menores valores da função perda e seus componentes, qual é a época e tempo gasto para obtenção para segunda rodada?

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Época de obtenção	Tempo até obtenção(m)
6	$2.919{\cdot}10^{-6}$	$2.866{\cdot}10^{-6}$	$1.182{\cdot}10^{-10}$	$2.130{\cdot}10^{-9}$	399700	78.599
7	$6.568{\cdot}10^{-6}$	$5.213{\cdot}10^{-6}$	$3.841{\cdot}10^{-10}$	$5.419{\cdot}10^{-8}$	395100	79.705
8	$7.503{\cdot}10^{-6}$	$6.897{\cdot}10^{-6}$	$4.287{\cdot}10^{-9}$	$2.408{\cdot}10^{-8}$	392100	70.694
9	$3.415{\cdot}10^{-5}$	$2.719{\cdot}10^{-5}$	$4.834{\cdot}10^{-10}$	$2.783{\cdot}10^{-7}$	397200	75.761

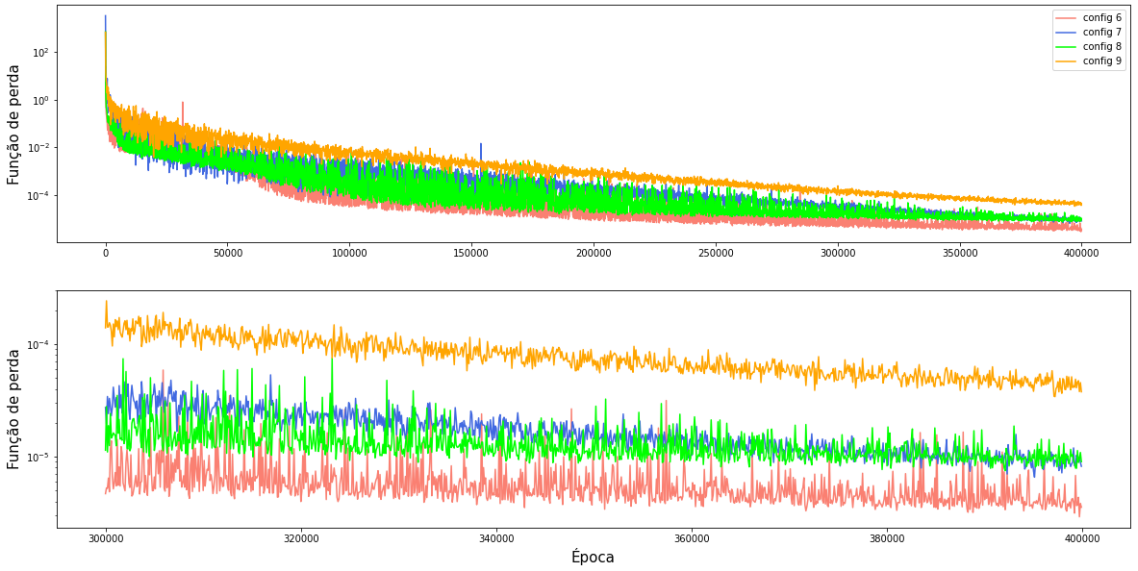

Figura 5.14: Valor da função de perda para todas as épocas (Superior); Foco nas últimas 100.000 épocas (Inferior). Referente à segunda rodada.

Nessa rodada o cientista pode perceber que ambas as configurações testadas

obtiveram menores valores para métrica R^2, se comparadas com suas versões com topologias mais simples da primeira rodada. Porém, nessa rodada os valores para a função de perda foram menores.

Conclusão do experimento

Com as consultas aplicadas à base de dados de proveniência, o cientista pode tirar conclusões importantes para o experimento. Ao aumentar a complexidade da PINN a função de perda tende a ser menor, porém, essa redução não acompanha a melhora da capacidade preditiva do modelo dada pela métrica R^2. Tal comportamento é um bom indicativo de realizar o experimento com uma nova abordagem para a função de perda. Caso o cientista busque implementar adaptações de pesos de maneira automática para os componentes da função de perda, essa transformação pode ser configurada utilizando a DNNProv para que sejam persistidas na base de dados de proveniência, assim como é feita com a adaptação do valor da taxa de aprendizado. Com isso, o cientista pode realizar consultas sobre essas transformações, como é feito na Tabela 5.12 para a taxa de aprendizado.

Tabela 5.12: Quais os valores de adaptação de taxa de aprendizado na época 300000 e o impacto para o treinamento da configuração 5?

Época	R^2	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Nova taxa de aprendizado
300000	0.451	$9.787 \cdot 10^{-5}$	$5.215 \cdot 10^{-5}$	$3.867 \cdot 10^{-9}$	$1.829 \cdot 10^{-6}$	$7.397 \cdot 10^{-4}$
300100	0.452	$1.372 \cdot 10^{-5}$	$5.300 \cdot 10^{-5}$	$5.461 \cdot 10^{-9}$	$3.369 \cdot 10^{-6}$	$7.390 \cdot 10^{-4}$

O cientista pode continuar realizando ajuste finos nos hiperparâmetros para buscar modelos de PINNs melhores para o seu problema. Nesse experimento, o melhor modelo foi o de configuração 5 com métrica $R^2 = 0.47$. A Figura 5.15 ilustra como a estimativa feita por essa PINN se aproxima da solução real do problema representada pela Figura 5.10.

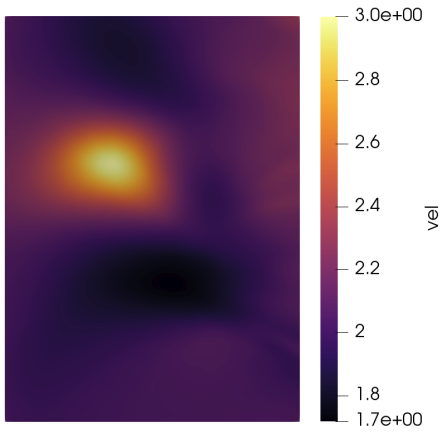

Figura 5.15: Predição do melhor modelo de PINN com acurácia $R^2 = 0.47$.

Caso o cientista não utilizasse o apoio da DNNProv, ele teria que realizar a gerência e coleta dos dados referentes às métricas de avaliação e hiperparâmetros de cada modelo utilizando arquivos log, que não forneceriam os benefícios da persistência dos dados feitas com proveniência da DNNProv. Também deveria definir uma organização para armazenar esses arquivos, que por sua vez não respeitaria um padrão pré-definido. Além do que, teria que programar um outro *script* para fazer a associação dos diferentes arquivos log referentes à cada modelo para depois realizar suas análises.

5.4 Estudo de caso: Proveniência com a PINN da equação de Poisson no DeepXDE

5.4.1 A Biblioteca DeepXDE e integração com DNNProv

A DeepXDE é uma biblioteca Python para resolução de PINNs. Foi projetada para servir tanto como uma ferramenta educacional para ser usada em sala de aula, quanto como uma ferramenta de pesquisa para resolução de problemas em ciência da computação e engenharia (LU *et al.*, 2021). Sua implementação é simples, o que justifica o seu ganho de popularidade com os cientistas desenvolvedores de PINNs. A DeepXDE foi desenvolvida para ser usada com cinco bibliotecas diferentes de treinamento no seu *back-end*, essas são: TensorFlow 1.x, TensorFlow 2.x, PyTorch, JAX, e PaddlePaddle. A DeepXDE não conta com uma solução nativa para a persistência de dados de proveniência durante o treinamento dos seus modelos, porém podemos utiliza-la integrada com a nossa extensão da DNNProv apresentada na Seção 4.1.

Para isso, basta criar um objeto de classe herdando a classe *deepxde.callbacks.Callback*. Nesse módulo, podemos utilizar métodos predefinidos que irão ser inciadas durante os passos chaves da execução do modelo. Por exemplo, em *on_train_begin*, as instruções de código serão executadas no início do treinamento do modelo; *on_train_end* no fim do treinamento; *on_epoch_end* no final de cada época. Com isso, podemos instrumentar a DNNProv para extrair e persistir todos os dados gerados no treinamento. O código a seguir é um exemplo de como essa instrumentação é feita para o nosso experimento.

```
class DNNProv_calls(deepxde.callbacks.Callback):
    def __init__(self):
        super().__init__()
        self.epoch = 0
        self.dataflow_tag = "deepxde"
        self.exec_tag = self.dataflow_tag +
            datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

```
        df = Dataflow(self.dataflow_tag,
        ['STR_OPTIMIZER_NAME',
        'NUM_LEARNING_RATE',
        'NUM_NUM_EPOCHS',
        'NUM_BATCH_SIZE',
        'STR_LAYERS',
        'NUM_WEIGHT_LR',
        'NUM_WEIGHT_LB',
        'NUM_WEIGHT_LD'],
        ['NUM_epoch',
        'NUM_time_elapsed',
        'NUM_LOSS',
        'NUM_LR_train',
        'NUM_LB_train',
        'NUM_LD_train',
        'NUM_Q_train_error',
        'NUM_U_train_error'])
        df.save()
    def on_train_begin(self):
        (...)
        self.t1 = Task(1, self.dataflow_tag, self.exec_tag, "TrainingModel")
        self.tf1_input = DataSet("iTrainingModel", [Element([opt_name,
                                                             learning_rate,
                                                             epoch,
                                                             batch_size,
                                                             layers_list,
                                                             weight_lr,
                                                             weight_lb,
                                                             weight_ld)])
        self.t1.add_dataset(self.tf1_input)
        self.t1.begin()
    def on_train_end(self):
        self.t1.end()
    def on_epoch_end(self):
        if(self.epoch % 10 == 0):
            (...)
            tf1_output = DataSet("oTrainingModel", [Element([epoch,
                                                             elapsed,
                                                             loss_train_value,
                                                             lr_value,
                                                             lb_value,
                                                             ld_value,
                                                             err_q_train,
                                                             err_u_train])])
            self.t1.add_dataset(tf1_output)
            self.t1.save()
        self.epoch += 1
```

Na inicialização da classe de *callback*, dada pelo método *__init__*, iniciamos a DNNProv e informamos as colunas referentes às entidades de entrada e saída ligadas à atividade do treinamento. No método *on_train_begin* estamos persistindo as informações referentes aos hiperparâmetros do modelo na entidade de entrada *iTrainingModel*. Já no método *on_epoch_end* persistimos as informações referentes às métricas de saída do treinamento na entidade *oTrainingModel*. Por fim, no *on_train_end*, finalizamos o fluxo de dados relacionado à atividade de treinamento.

Com o objeto do tipo *callback* criado, basta informá-lo no argumento de *callback* que o método de treinamento do modelo já espera.

```
(...)
model.train(iterations,callbacks=[DNNProv_calls()],batch_size)
```

Seguindo essa estrutura, podemos utilizar a DNNProv em conjunto com a DeepXDE para qualquer tipo de aplicação e em qualquer um dos seus cinco *back-ends* de desenvolvimento.

O cientista pode construir o objeto do tipo *callback* fazendo a persistência dos dados de cada modelo utilizando por exemplo a biblioteca Pandas. Porém, a organização dos arquivos ficaria a cargo do cientista e não possuirá um padrão pré-definido. Além do que também teria que criar um novo *script* para fazer a associação dos arquivos referentes aos dados relacionados aos hiperparâmetros de entrada e métricas de saída de cada modelo para fazer suas análises. Utilizando a DNNProv o cientista vai ter um padrão para o armazenamento dos dados do seu experimento utilizando proveniência, e basta fazer as associações entre as entidades de entrada e saída da atividade de treinamento para realizar suas análises comparativas de modelos.

5.4.2 Experimento

Para esse experimento usamos a nossa extensão da DNNProv para coleta de dados de proveniência para um dos problemas exemplo fornecidos pela DeepXDE, o problema inverso para a equação de Poisson com campo de força desconhecido[3]. Esse tipo de equação é frequentemente encontrada em problemas de Física e Engenharia. A equação é em uma dimensão e é da seguinte forma:

$$\frac{d^2u(x)}{dx^2} = q(x), x \in [-1, 1]$$

Com as condições de contorno de Dirichlet:

$$u(-1) = u(1) = 0$$

[3]https://deepxde.readthedocs.io/en/latest/demos/pinn_inverse/elliptic.inverse.field.html

Nesse cenário $u(x)$ e $q(x)$ são desconhecidos. Para resolver o problema, temos o valor de $u(x)$ em 100 pontos. A solução de referência é $u(x) = sin(\pi x), q(x) = -\pi^2 sin(\pi x)$. Usamos duas redes, uma para treinar para $u(x)$ e a outra para treinar para $q(x)$ em 50000 épocas, persistindo os dados de saída a cada intervalo de 10 épocas. Nesse experimento foi utilizado como métrica de avaliação o erro relativo L2.

Para realizar a análise comparativa dos modelos, foram utilizadas as mesmas consultas à nossa base de proveniência apresentada no Experimento 5.3. Avaliamos três abordagens para os pesos ligados à função de perda da PINN, representado pela Equação 2.2, as configurações do modelo são apresentadas na Tabela 5.13, onde os hiperparâmetros referentes à rede neural são comuns para ambas as redes responsáveis pelas estimativas de $u(x)$ e $q(x)$.

Tabela 5.13: Quais são as configurações das PINNs treinadas?

ID	Otimizador	Ativação	N.º de Neurônios	N.º de Camadas Intermediárias	Taxa de Aprendizado	Pesos da funcão de perda (w_R, w_{BC}, w_D)
1	Adam	tanh	20	4	$1{\cdot}10^{-4}$	(10,100,1000)
2	Adam	tanh	20	4	$1{\cdot}10^{-4}$	(15,150,1500)
3	Adam	tanh	20	4	$1{\cdot}10^{-4}$	(20,200,2000)

Tabela 5.14: Quais são os menores valores do erro relativo L2 para as estimativas da função u(x), qual é a época e tempo gasto para obtenção?

ID	Mínimo erro relativo L2	Época de obtenção	Tempo até obtenção(s)	Média 10 épocas(s)
3	$1.98{\cdot}10^{-4}$	49590	83.064	0.017
2	$2.89{\cdot}10^{-4}$	49990	83.778	0.017
1	$2.97{\cdot}10^{-4}$	49860	88.809	0.018

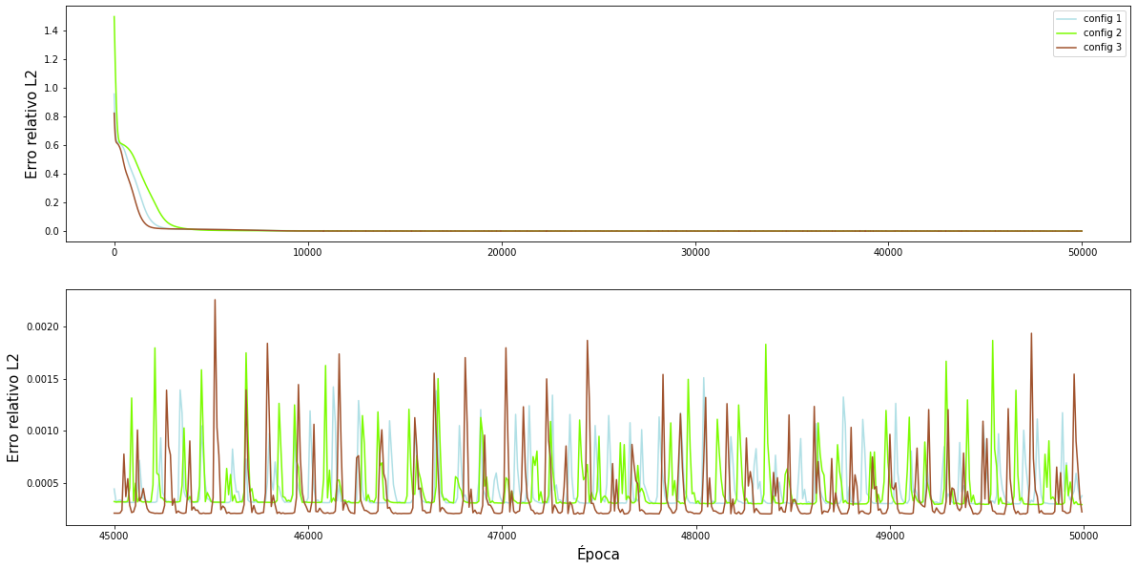

Figura 5.16: Valor do erro relativo L2 para as estimativas da função u(x) para todas as épocas (Superior); Foco nas últimas 5.000 épocas (Inferior).

Tabela 5.15: Quais são os menores valores da função perda e seus componentes, qual é a época e tempo gasto?

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Época de obtenção	Tempo até obtenção(s)
1	$5.384{\cdot}10^{-5}$	$9.980{\cdot}10^{-7}$	$1.975{\cdot}10^{9}$	$4.366{\cdot}10^{-8}$	49860	88.809
3	$6.359{\cdot}10^{-5}$	$1.017{\cdot}10^{-6}$	$2.037{\cdot}10^{8}$	$1.959{\cdot}10^{-8}$	49830	83.466
2	$6.751{\cdot}10^{-5}$	$1.666{\cdot}10^{-7}$	$1.166{\cdot}10^{8}$	$4.218{\cdot}10^{-8}$	49990	83.778

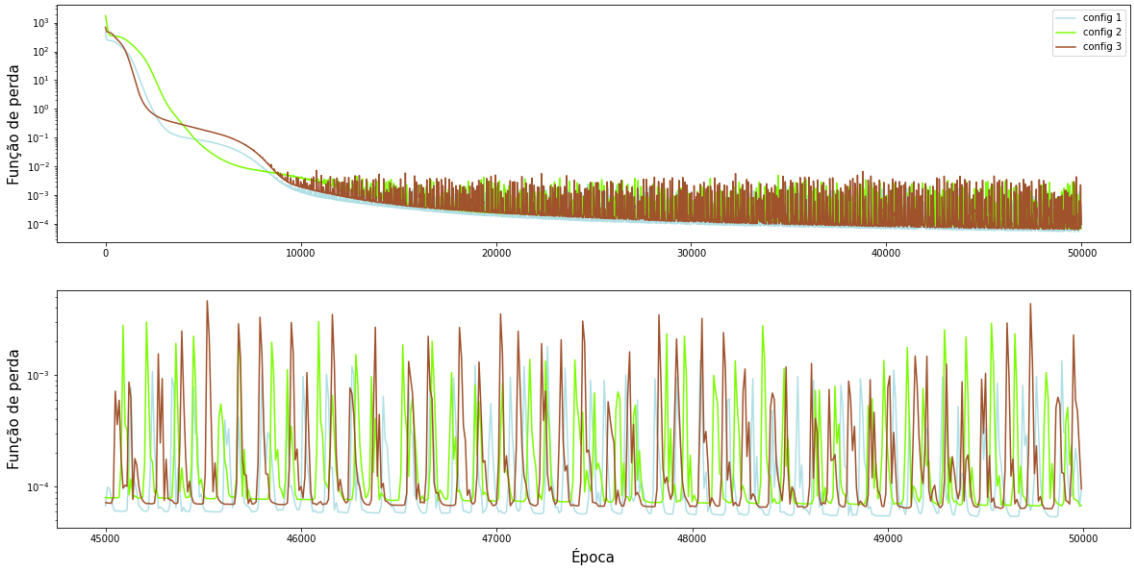

Figura 5.17: Valor da função de perda para todas as épocas (Superior); Foco nas últimas 5.000 épocas (Inferior).

Como é um experimento simples, os tempos de execução para cada configuração nas 50000 épocas foram de no máximo 90 segundos. O resultado da melhor métrica do erro relativo L2 e o menor valor da função de perda não apresentaram uma grande discrepância entre os modelos. Levando em consideração a métrica do erro relativo L2, o melhor modelo foi o de configuração 3. A estimativa desse modelo comparado com o valor real são ilustrados na Figura 5.18.

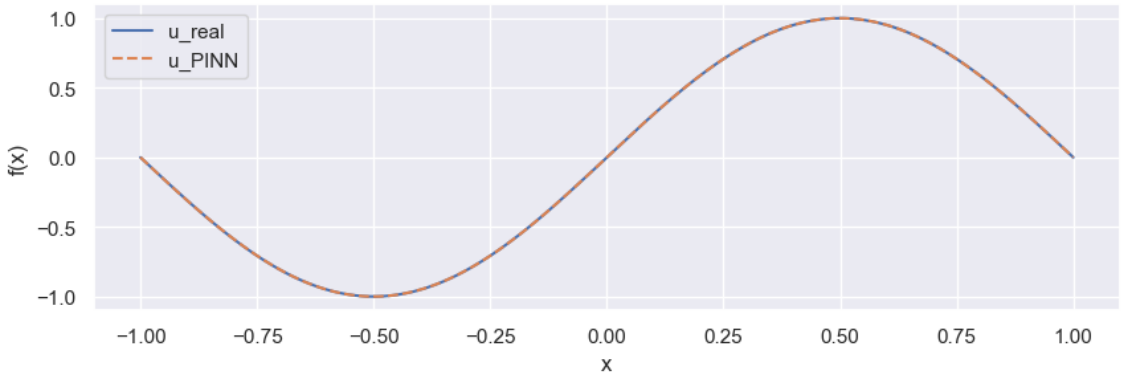

Figura 5.18: Comparação entre a função real de u e a função estimada pela nossa melhor PINN.

Apesar de ser um exemplo simples, a persistência de dados com proveniência utilizando a DNNProv pode ser aplicada para qualquer tipo de experimento realizado no DeepXDE, desde que a instrumentação seja feita de maneira correta, como foi mostrado anteriormente.

Capítulo 6

Conclusão

Ferramentas para desenvolvimento de DNNs possibilitam a persistência dos dados de treinamento via exportação das informações dos modelos em arquivo log, geralmente em formato CSV. Apesar de ser uma alternativa, esse arquivo não fornece os benefícios dos dados de treinamento armazenados com auxílio de proveniência em um sistema de banco de dados, que já possibilita consultas em tempo de execução nos dados estruturados e análises utilizando caminho de derivação.

Essa abordagem de análise de dados escalares via CSV se torna inviável considerando o número de configurações que vem sendo explorados, em especial as PINNs que aumentam os parâmetros de configuração das DNNs. Como mostrado, é possível analisar a proveniência de modelos e gerar resultados importantes para apoiar o usuário na escolha da melhor configuração de PINNs utilizando a base de dados de proveniência. Da mesma forma, as bases de proveniência podem auxiliar na escolha dos pesos da função de perda da PINN.

A biblioteca de proveniência DNNProv mostrou ser uma boa solução para esse tipo de abordagem, dado que a mesma facilita a persistência dos dados de treinamento em SGBD utilizando o modelo de proveniência recomendado pelo W3C PROV. Porém a mesma é genérica para qualquer tipo de DNN. Dessa maneira, o usuário desenvolvedor de PINNs necessita instrumentar no código de treinamento manualmente quais dados do seu modelo devem ser persistidos, podendo então dificultar na usabilidade dessa abordagem.

Com a extensão realizada nesta dissertação, o uso da DNNProv para PINNs foi facilitado e mostramos que a abordagem proposta pode ser usada em diferentes ambientes pelos cientista de PINNs. As consultas realizadas mostram o apoio dado ao cientista e o aumento de confiabilidade no experimento por meio do armazenamento dos dados de proveniência. É importante destacar que para obter respostas a essas consultas sem a DNNProv, o cientista teria que instrumentar o código, gerenciar a coleta de dados, definir uma organização para os dados e escolher uma forma de armazenamento para a persistência e um acesso aos dados para a consulta.

Para trabalhos futuros levamos em consideração a especialização da DNNProv

para aplicações de PINNs, levando como exemplo a especialização feita pela Keras-Prov. Comparando os nossos experimentos, a instrumentação da Keras-Prov foi realizada de maneira consideravelmente mais fácil se comparado com as feitas utilizando a DNNProv. Porém diferente da Keras-Prov, a especialização para PINNs não deve estar atrelada a um *framework* específico de treinamento. Atualmente há uma variedade de ferramentas que oferecem apoio para treinamento de PINNs, fixar a sua utilização em somente uma ou algumas dessas, pode limitar a sua adoção por parte dos cientistas. Além disso, acompanhar o amadurecimento das soluções em AutoML para PINNs e propor uma integração das análises dos dados de proveniência ao longo do treinamento e realizar adaptações junto aos algoritmos de AutoML.

Também pretendemos avaliar os benefícios da gerência de dados de proveniência em outros tipos de algoritmos de DNNs ligados à Física, como por exemplo *Variational Physics-Informed Neural Networks*, *Deep Operator Network*, *Physics-informed Deep Operator Network*, modelos substitutivos baseados em redes neurais codificadoras-decodificadoras, entre outros. O objetivo seria avaliar se a extensão da DNNProv utilizada para fazer a captura dos dados de proveniência em PINNs desta dissertação é suficiente para apoiar os cientistas desenvolvedores nesses algoritmos de desenvolvimento citados, ou se é necessário realizar outras modificações na biblioteca ou adotar uma diferente solução.

Referências Bibliográficas

BAKER, M., 2016, "1,500 scientists lift the lid on reproducibility", *Nature*, v. 533, n. 7604.

BISCHOF, R., KRAUS, M., 2021, "Multi-Objective Loss Balancing for Physics-Informed Deep Learning", (10). doi: 10.13140/RG.2.2.20057.24169.

CAMPOS, V. S., 2018, "DfA-lib-Python: Uma biblioteca para a extração de dados científicos usando a DfAnalyzer", .

CUOMO, S., COLA, V. S. D., GIAMPAOLO, F., et al., 2022, "Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next", *Springer Journal of Scientific Computing*. doi: https://doi.org/10.1007/s10915-022-01939-z.

DE OLIVEIRA, L. S., SILVA, R. M., KUNSTMANN, L., et al., 2022, "Dados de proveniência para redes neurais guiadas pela Física: o caso da equação eikonal". In: *Anais do XXXVII Simpósio Brasileiro de Bancos de Dados*, pp. 373–378. SBC.

DEBNATH, L., 2012, "First-Order Nonlinear Equations and Their Applications". In: *Nonlinear Partial Differential Equations for Scientists and Engineers*, pp. 227–256, Boston, Birkhäuser Boston. ISBN: 978-0-8176-8265-1. doi: 10.1007/978-0-8176-8265-1_4. Disponível em: <`https://doi.org/10.1007/978-0-8176-8265-1_4`>.

DENG, J., DONG, W., SOCHER, R., et al., 2009, "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee.

FAGNAN, K., NASHED, Y., PERDUE, G., et al., 2019, *Data and models: a framework for advancing AI in science*. Relatório técnico, USDOE Office of Science (SC)(United States).

FREIRE, J., KOOP, D., SANTOS, E., et al., 2008, "Provenance for computational tasks: A survey", *Computing in science & engineering*, v. 10, n. 3, pp. 11–21.

GHARIBI, G., WALUNJ, V., RELLA, S., et al., 2019, "Modelkb: towards automated management of the modeling lifecycle in deep learning". In: *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pp. 28–34. IEEE.

GIL, Y., MILES, S., BELHAJJAME, K., et al., 2013, "Prov model primer", *W3C Working Group Note*.

GOODFELLOW, I., BENGIO, Y., COURVILLE, A., 2016, *Deep Learning.* ", MIT Press. http://www.deeplearningbook.org.

HAGHIGHAT, E., JUANES, R., 2021, "SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks", *Computer Methods in Applied Mechanics and Engineering*, v. 373, pp. 113552.

HAN, R., BYNA, S., TANG, H., et al., 2022, "PROV-IO: An I/O-Centric Provenance Framework for Scientific Data on HPC Systems". In: *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, pp. 213–226.

HENNIGH, O., NARASIMHAN, S., NABIAN, M. A., et al., 2021, "NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework". In: *International conference on computational science*, pp. 447–461. Springer.

HUTSON, M., 2018. "Artificial intelligence faces reproducibility crisis". .

JIN, X., CAI, S., LI, H., et al., 2021, "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations", *Journal of Computational Physics*, v. 426, pp. 109951. ISSN: 0021-9991. doi: https://doi.org/10.1016/j.jcp.2020.109951. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999120307257>.

KARMAKER, S. K., HASSAN, M. M., SMITH, M. J., et al., 2021, "AutoML to Date and Beyond: Challenges and Opportunities", *ACM Comput. Surv.*, v. 54, n. 8 (oct), pp. 1–36. ISSN: 0360-0300. doi: 10.1145/3470918. Disponível em: <https://doi.org/10.1145/3470918>.

KIDGER, P., LYONS, T., 2020, "Universal approximation with deep narrow networks". In: *Conference on learning theory*, pp. 2306–2327.

KITCHENHAM, B., 2004, "Procedures for performing systematic reviews", *Keele, UK, Keele University*, v. 33, n. 2004, pp. 1–26.

KITCHENHAM, B., CHARTERS, S., 2007. "Guidelines for performing systematic literature reviews in software engineering". .

KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., 2012, "ImageNet Classification with Deep Convolutional Neural Networks". In: Pereira, F., Burges, C., Bottou, L., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 25. Curran Associates, Inc. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

KUMAR, A., NAKANDALA, S., ZHANG, Y., et al., 2021, "Cerebro: A Layered Data Platform for Scalable Deep Learning". In: *11th Annual Conference on Innovative Data Systems Research (CIDR'21)*.

KUNSTMANN, L., PINA, D., SILVA, F., et al., 2021, "Online Deep Learning Hyperparameter Tuning based on Provenance Analysis", *Journal of Information and Data Management*, v. 12, n. 5, pp. 396–414.

LECUN, Y., BENGIO, Y., HINTON, G., 2015, "Deep learning", *nature*, v. 521, n. 7553, pp. 436–444.

LI, L., NAKANDALA, S., KUMAR, A., 2021, "Intermittent human-in-the-loop model selection using cerebro: a demonstration", *Proceedings of the VLDB Endowment*, v. 14, n. 12, pp. 2687–2690.

LU, L., MENG, X., MAO, Z., et al., 2021, "DeepXDE: A deep learning library for solving differential equations", *SIAM Review*, v. 63, n. 1, pp. 208–228. doi: 10.1137/19M1274067.

MIAO, H., LI, A., DAVIS, L. S., et al., 2017, "Towards unified data and lifecycle management for deep learning". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 571–582. IEEE.

MISSIER, P., 2016, "The lifecycle of provenance metadata and its associated challenges and opportunities", *Building Trust in Information: Perspectives on the Frontiers of Provenance*, pp. 127–137.

MISSIER, P., BELHAJJAME, K., CHENEY, J., 2013, "The W3C PROV family of specifications for modelling provenance metadata". In: *Proceedings of the 16th International Conference on Extending Database Technology*, pp. 773–776.

MOREAU, L., GROTH, P., 2013, *Provenance: An Introduction to PROV*. Synthesis Lectures on the Semantic Web: Theory and Technology. ", Morgan & Claypool Publishers. ISBN: 9781627052214. doi: 10.2200/S00528ED1V01Y201308WBE007. Disponível em: <https://doi.org/10.2200/S00528ED1V01Y201308WBE007>.

MOREAU, L., (EDS.), P. M., BELHAJJAME, K., et al., 2013, "Prov-dm: The prov data model", *W3C Recommendation*.

MUSTAFA, T. A., KÖNIG-RIES, B., SAMUEL, S., 2023, "MLProvCodeGen: A Tool for Provenance Data Input and Capture of Customizable Machine Learning Scripts", *BTW 2023*.

NILSBACK, M.-E., ZISSERMAN, A., 2006, "A Visual Vocabulary for Flower Classification". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, v. 2, pp. 1447–1454. doi: 10.1109/CVPR.2006.42.

NOURI, A., DAVIS, P. E., SUBEDI, P., et al., 2021, "Exploring the role of machine learning in scientific workflows: Opportunities and challenges", *arXiv preprint arXiv:2110.13999*.

PASZKE, A., GROSS, S., CHINTALA, S., et al., 2017, "Automatic differentiation in pytorch", *OpenReview*. Disponível em: <https://openreview.net/forum?id=BJJsrmfCZ>.

PINA, D., KUNSTMANN, L., DE OLIVEIRA, D., et al., 2020, "Provenance Supporting Hyperparameter Analysis in Deep Neural Networks". In: *Provenance and Annotation of Data and Processes*, Springer, pp. 20–38.

PINA, D., KUNSTMANN, L., DE OLIVEIRA, D., et al., 2021, "Provenance supporting hyperparameter analysis in deep neural networks". In: *Provenance and Annotation of Data and Processes: 8th and 9th International Provenance and Annotation Workshop, IPAW 2020+ IPAW 2021, Virtual Event, July 19–22, 2021, Proceedings 8*, pp. 20–38. Springer.

RAISSI, M., PERDIKARIS, P., KARNIADAKIS, G. E., 2019, "Physics-informed neural networks: A deep learning framework for solving forward and in-

verse problems involving nonlinear partial differential equations", *Journal of Computational physics*, v. 378, pp. 686–707.

SAMUEL, S., KÖNIG-RIES, B., 2021, "Understanding experiments and research practices for reproducibility: an exploratory study", *PeerJ*, v. 9, pp. e11140.

SCHELTER, S., BOESE, J.-H., KIRSCHNICK, J., et al., 2017, "Automatically tracking metadata and provenance of machine learning experiments", .

SCHLEGEL, M., SATTLER, K.-U., 2023, "Management of Machine Learning Lifecycle Artifacts: A Survey", *ACM SIGMOD Record*, v. 51, n. 4, pp. 18–35.

SILVA, R., PINA, D., KUNSTMANN, L., et al., 2021, "Capturing Provenance to Improve the Model Training of PINNs: first hand-on experiences with Grid5000". In: *42nd CILAMCE*, pp. 1–7.

SILVA, V., DE OLIVEIRA, D., VALDURIEZ, P., et al., 2018, "DfAnalyzer: runtime dataflow analysis of scientific applications using provenance", *Proceedings of the VLDB Endowment*, v. 11, n. 12, pp. 2082–2085.

SOUZA, A., 2023, *A study on domain treatment methods in physics-informed neural networks*. Tese de Mestrado, Programa de Engenharia Civil, Universidade Federal do Rio de Janeiro.

SOUZA, R., AZEVEDO, L., LOURENÇO, V., et al., 2019a, "Provenance Data in the Machine Learning Lifecycle in Computational Science and Engineering". In: *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pp. 1–10, a. doi: 10.1109/WORKS49585.2019.00006.

SOUZA, R., AZEVEDO, L., THIAGO, R., et al., 2019b, "Efficient Runtime Capture of Multiworkflow Data Using Provenance". In: *2019 15th International Conference on eScience (eScience)*, pp. 359–368, b. doi: 10.1109/eScience.2019.00047.

SOUZA, R., AZEVEDO, L. G., LOURENÇO, V., et al., 2022, "Workflow provenance in the lifecycle of scientific machine learning", *Concurrency and Computation: Practice and Experience*, v. 34, n. 14, pp. e6544.

STEVENS, R., TAYLOR, V., NICHOLS, J., et al., 2020, *Ai for science: Report on the department of energy (doe) town halls on artificial intelligence (ai) for science*. Relatório técnico, Argonne National Lab.(ANL), Argonne, IL (United States).

TSAY, J., MUMMERT, T., BOBROFF, N., et al., 2018, "Runway: machine learning model experiment management tool". In: *Conference on systems and machine learning (sysML)*.

VARTAK, M., SUBRAMANYAM, H., LEE, W.-E., et al., 2016, "ModelDB: a system for machine learning model management". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pp. 1–3.

WANG, D., WEISZ, J. D., MULLER, M., et al., 2019, "Human-AI collaboration in data science: Exploring data scientists' perceptions of automated AI", *Proceedings of the ACM on Human-Computer Interaction*, v. 3, n. CSCW, pp. 1–24.

WOZNIAK, J. M., LIU, Z., VESCOVI, R., et al., 2022, "Braid-DB: Toward AI-driven science with machine learning provenance". In: *Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation: 21st Smoky Mountains Computational Sciences and Engineering, SMC 2021, Virtual Event, October 18-20, 2021, Revised Selected Papers*, Springer, pp. 247–261.