Optimization of Physics-Informed Neural Network for Solving Partial Differential Equation with Uncertain Parameters

Sumanth Nethi
Dept of Aeronautics and Astronautics
Stanford University
Stanford, CA
nsumanth@stanford.edu

***Abstract*—In this project, I employed Physics-Informed Neural Networks (PINNs) as an effective tool for the data-driven discovery of the Burgers Equation, a non-linear Partial Differential Equation (PDE), with uncertain parameters. The Burgers Equation was selected for its distinctiveness, facilitating the design of smaller neural network architectures, and enabling the computation of exact solutions through the Finite Difference Method (FDM) Solver. Through this project, I intend to demonstrate the significant potential of PINNs in providing efficient and accurate solutions for non-linear PDEs, showcasing its considerable promise in the broader realm of computational physics by comparing it to regular Artificial Neural Networks (ANNs). My results highlight promising avenues for further research, implying its broader applicability to a range of complex physical systems that can be modelled by non-linear PDEs.**

I. Introduction

As the fields of computational physics and machine learning continue to intersect and evolve, innovative methodologies are being born. Physics-Informed Neural Networks (PINNs) stand as one such groundbreaking approach that ingeniously merges the computational capability of neural networks with the rigidity and accuracy of physics-based models.

Neural networks, the cornerstone of modern machine learning, are incredibly flexible function approximators. Their ability to learn from complex and high-dimensional datasets has led to transformative advancements across numerous fields. However, conventional neural networks, or Artificial Neural Networks (ANNs), have limitations when it comes to solving scientific computing problems, particularly those involving the solution of Partial Differential Equations (PDEs).

Traditional ANNs are purely data-driven, meaning they lack the ability to incorporate known physical laws or principles into the learning process. As a result, while they can achieve impressive predictive performance when provided with ample high-quality data, they can struggle in situations where the data are sparse, noisy, or uncertain. Moreover, ANNs might produce results that, while numerically accurate according to their training data, could violate fundamental laws of physics, leading to non-physical and thus untrustworthy predictions. For example, mass cannot be negative but however, the ANN could output a negative number.

This is where Physics-Informed Neural Networks (PINNs) step in, bridging this gap by introducing physical laws directly into the loss function during the learning process. By doing so, PINNs inherently ensure that the solutions they generate adhere to the underlying physical principles described by the governing equations. This feature allows PINNs to produce more reliable and accurate results compared to traditional ANNs, especially when dealing with sparse or noisy data.

In this project, I explore the capability of PINNs through a data-driven discovery of a non-linear PDE: the Burgers Equation. The Burgers Equation, a simplified form of the Navier-Stokes equation, is a model problem in the study of turbulence and non-linear wave phenomena. By infusing the governing physical laws of this equation into a PINN, I aim to discover the equation from noisy data with uncertain parameters, highlighting the efficacy and robustness of the PINN approach.

My objective is to showcase the significant potential of PINNs in providing efficient and accurate solutions for non-linear PDEs. Through a comprehensive analysis of my approach - the generation of training datasets, the design and training of neural networks, and the evaluation of the model's performance (predicted equation or uncertain parameters) - I hope to contribute to the broader idea surrounding PINNs, encouraging their continued exploration and application in computational physics.

II. Literature Review

Physics-Informed Neural Networks (PINNs) have garnered significant attention in recent years, particularly for their potential in solving forward and inverse problems involving non-linear Partial Differential Equations (PDEs). This concept was prominently introduced and detailed by Raissi, Perdikaris, and Karniadakis in their seminal two-part work [1], [2]. In these papers, the authors explored data-driven solutions of non-linear PDEs, and extended the PINN framework to data-driven discovery of these equations, setting the groundwork for further research in this area.

Several researchers have then focused on creating specialized software libraries that aid in solving differential equations using deep learning frameworks. For instance, Lu et al. developed the DeepXDE library [3], Koryagin et al. proposed the PyDEns framework [6], and Chen et al. introduced the NeuroDiffEq package [7]. Each of these tools provides unique functionalities to enhance the implementation and usability of PINNs and similar deep learning methods for solving differential equations.

At the intersection of PINNs and Hypernetworks, Belbute-Peres et al. introduced HyperPINNs [4], a novel concept where a hypernetwork is utilized to generate the parameters of a PINN, adding another level of complexity and potential to the PINN framework. Similarly, Rackauckas et al. expanded on the PINN methodology, introducing the concept of Universal Differential Equations for scientific machine learning [8]. This method combines traditional differential equations and machine learning to solve scientific problems.

As the field has evolved, more sophisticated PINN variants have been introduced, catering to a range of unique problem sets. A noteworthy advancement in this direction is the development of Adversarial and Bayesian PINNs [9], [10], which incorporate uncertainty quantification in the learning process. This addition allows the model to learn from uncertain and noisy data, significantly broadening the scope and applicability of PINNs which is majorly focused in this project.

Literature also demonstrates the potential of PINNs in handling Small Data problems. Grigo and Koutsourelakis proposed a physics-aware, probabilistic machine learning framework for coarse-graining high-dimensional systems in the Small Data regime [11]. Such studies hint at the exciting possibilities of employing PINNs in scenarios where data availability is a major challenge.

Finally, in terms of theoretical developments, Mishra and Molinaro provided estimates on the generalization error of PINNs for approximating a class of inverse problems for PDEs [12], contributing towards a theoretical understanding of the PINN approach and its limitations. This theoretical perspective is essential to guide future research and applications of PINNs.

In summary, the field of PINNs has witnessed rapid advancements, both in theory and applications, since its inception in 2017. This project draws upon these developments, employing the PINN framework to solve the Burgers Equation, a classical problem in the study of non-linear wave phenomena, and thereby contributing my share to this exciting field.

III. Problem Setup

The primary focus of this project is the data-driven discovery of partial differential equations (PDEs), specifically the one-dimensional Burgers' equation. I shall explore this within the framework of parameterized and non-linear PDEs, generally represented as $u_t + N[u;\lambda] = 0$, where $x \in \Omega, t \in [0,T]$. Here, $u(t,x)$ represents the latent (hidden) solution, $N[;\lambda]$ is a nonlinear operator parameterized by λ, and Ω is a subset of R^D. This configuration encapsulates a broad spectrum of problems in mathematical physics, including conservation laws, diffusion processes, advection-diffusion-reaction systems, and kinetic equations.

In the context of this project, I specifically focus on the one-dimensional Burgers' equation. This PDE finds widespread application in various fields of applied mathematics, including fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow. It is a fundamental PDE that can be derived from the Navier-Stokes equations for the velocity field by neglecting the pressure gradient term. For small viscosity parameter values, the Burgers' equation can lead to shock formation, a phenomenon that is notoriously challenging to resolve using traditional numerical methods.

The Burgers' equation is given in the form $N[u;\lambda] = \lambda_1 u u_x - \lambda_2 u_{xx}$, with $\lambda = (\lambda_1, \lambda_2)$. Here, the subscripts denote partial differentiation with respect to either time or space. Thus, in one spatial dimension, the Burgers' equation reads as:

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0 \tag{1}$$

The objective now is to determine the parameters λ that can best explain the observed data, given a small set of scattered and potentially noisy observations of the hidden state $u(t,x)$.

IV. Approach

The problem of data-driven discovery of partial differential equations (PDEs) is tackled using a continuous time model. The Burgers' equation of focus is defined as

$$u_t + u u_x - \frac{0.01}{\pi} u_{xx} = 0 \tag{2}$$

with boundary conditions and initial conditions specified as:

$$x \in [-1,1], t \in [0,1] \tag{3}$$

$$u(0,x) = -\sin(\pi x), u(t,-1) = u(t,1) = 0 \tag{4}$$

These conditions correspond to the parameters $\lambda_1 = 1.0$ and $\lambda_2 = \frac{0.01}{\pi}$ in the general non-linear operator.

The first step involves generating scattered and noisy measurements across the entire spatio-temporal domain for this fixed Burgers' equation using Finite Difference Method (FDM) solvers, achieving adequate accuracy. A Physics-Informed Neural Network (PINN) is then employed to train the model. This necessitates a modification to the traditional loss function in order to incorporate physical constraints.

Let $f(t,x) := u_t + \lambda_1 u u_x - \lambda_2 u_{xx}$, and approximate $u(t,x)$ by a deep neural network, resulting in the physics-informed neural network $f(t,x)$. The shared parameters of $u(t,x)$ and $f(t,x)$, along with the parameters $\lambda = (\lambda_1, \lambda_2)$ of the differential operator, can be learned by minimizing the mean squared error loss of the modified loss function:

$$MSE = MSE_u + MSE_f, \tag{5}$$

where:

$$MSE_u = \frac{1}{N} \sum_{i=1}^{N} |u(t_u^i, x_u^i) - u_i|^2, \tag{6}$$

and

$$MSE_f = \frac{1}{N}\sum_{i=1}^{N} |f(t_u^i, x_u^i)|^2. \tag{7}$$

Here, $\{t_u^i, x_u^i, u_i\}_{i=1}^N$ denote the training data on $u(t,x)$. The loss MSE_u corresponds to the training data on $u(t,x)$, while MSE_f enforces the structure imposed by the Burgers equation at a finite set of collocation points.

A training dataset was created by randomly generating $N = 2500$ points across the entire spatio-temporal domain from the exact solution. This dataset was used to train a nine-layer deep neural network with 25 neurons per hidden layer by minimizing the mean square error of the new loss function as shown in equation (5) using the L-BFGS optimizer as well as the Adam optimizer. The network is then calibrated to predict the unknown parameters $\lambda = (\lambda_1, \lambda_2)$ defining the underlying dynamics. A similar training process is performed for a regular Artificial Neural Network (ANN) using the Adam optimizer, without modification of the loss function. The prediction performances of the PINN and the ANN are then compared.

V. Optimization Techniques

In this project, two main optimization techniques are employed for training the Physics-Informed Neural Networks (PINNs): Adam, a gradient descent-based method, and Limited-memory BFGS (L-BFGS), a quasi-Newton method.

A. Gradient Descent-Based Methods

Gradient descent-based methods are a go-to in training neural networks. The fundamental idea is to minimize the loss function iteratively by adjusting the network parameters in the direction of the steepest descent - the negative gradient of the loss function with respect to the parameters.

1) Adam Optimizer: Adam, which stands for Adaptive Moment Estimation, is a gradient-based optimization algorithm that combines the ideas of Momentum and RMSprop to create an optimizer that is computationally efficient and performs well in practice.

The update rule for Adam is given by maintaining and updating estimates of the first and second moments of the gradient. The general process in the k^{th} iteration is as follows:

- Compute the gradient $g(k)$ at the current point.
- Update the biased first moment estimate: $v^{k+1} = \gamma_v \cdot v^k + (1-\gamma_v)\cdot g^k$.
- Update the biased second raw moment estimate: $s^{k+1} = \gamma_s \cdot s^k + (1-\gamma_s)\cdot g(k) \odot g(k)$.
- Correct bias in first moment: $\hat{v}^{k+1} = v^{k+1}/(1-\gamma v^k)$.
- Correct bias in second moment: $\hat{s}(k+1) = s(k+1)/(1-\gamma s^k)$.
- Update parameters: $x^{k+1} = x^k - \alpha \cdot \hat{v}^{k+1}/(\sqrt{\hat{s}^{k+1}} + \epsilon)$.

In these equations, $\odot$ denotes the element-wise multiplication operation. The parameters γ_v, γ_s, and α are hyperparameters of the Adam optimizer, where γ_v and γ_s are the exponential decay rates for the moment estimates, α is the step size, and ϵ is a small number to prevent any division by zero. In practice, these are often set to $\gamma_v = 0.9$, $\gamma_s = 0.999$, and $\epsilon = 10^{-8}$.

B. Quasi-Newton Methods

Quasi-Newton methods are a class of optimization techniques that construct an approximation to the Hessian matrix, the second derivative of the loss function. By taking into account second-order information, these methods can potentially achieve faster convergence compared to first-order methods.

1) Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS): Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) method is a modification of the standard BFGS algorithm, designed to handle large-scale optimization problems that exceed the computer memory. Instead of storing the full BFGS Hessian matrix, L-BFGS maintains a limited history of the past gradient vectors from which it calculates an approximation to the inverse Hessian matrix.

L-BFGS maintains a history of the last m instances for $\boldsymbol{\delta}$ and $\boldsymbol{\gamma}$, as opposed to storing the complete inverse Hessian. Here, $i = 1$ corresponds to the oldest instance, and $i = m$ denotes the most recent one.

$$\boldsymbol{\gamma}^{(k+1)} \equiv \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}$$
$$\boldsymbol{\delta}^{(k+1)} \equiv \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$$

The procedure for determining the descent direction $\mathbf{d}$ at position $\mathbf{x}$ starts by calculating $\mathbf{q}^{(m)} = \nabla f(\mathbf{x})$. The remaining vectors $\mathbf{q}^{(i)}$, for i ranging from $m-1$ to 1, are calculated using

$$\mathbf{q}^{(i)} = \mathbf{q}^{(i+1)} - \frac{\left(\boldsymbol{\delta}^{(i+1)}\right)^\top \mathbf{q}^{(i+1)}}{\left(\boldsymbol{\gamma}^{(i+1)}\right)^\top \boldsymbol{\delta}^{(i+1)}}\boldsymbol{\gamma}^{(i+1)} \tag{8}$$

These vectors are then employed to compute another $m+1$ vectors, starting with

$$\mathbf{z}^{(0)} = \frac{\boldsymbol{\gamma}^{(m)} \odot \boldsymbol{\delta}^{(m)} \odot \mathbf{q}^{(m)}}{\left(\boldsymbol{\gamma}^{(m)}\right)^\top \boldsymbol{\gamma}^{(m)}} \tag{9}$$

$$\mathbf{z}^{(i)} = \mathbf{z}^{(i-1)} + \boldsymbol{\delta}^{(i-1)}\left(\frac{\left(\delta^{(i-1)}\right)^\top \mathbf{q}^{(i-1)}}{\left(\boldsymbol{\gamma}^{(i-1)}\right)^\top \boldsymbol{\delta}^{(i-1)}} - \frac{\left(\boldsymbol{\gamma}^{(i-1)}\right)^\top \mathbf{z}^{(i-1)}}{\left(\boldsymbol{\gamma}^{(i-1)}\right)^\top \boldsymbol{\delta}^{(i-1)}}\right)$$

The direction of descent, $\mathbf{d}$, is given by $\mathbf{d} = -\mathbf{z}^{(m)}$. For the purpose of minimization, it is required that the inverse Hessian, $\mathbf{Q}$, stays positive definite. The initial Hessian is typically established as:

$$\mathbf{Q}^{(1)} = \frac{\boldsymbol{\gamma}^{(1)}\left(\boldsymbol{\delta}^{(1)}\right)^\top}{\left(\boldsymbol{\gamma}^{(1)}\right)^\top \boldsymbol{\gamma}^{(1)}} \tag{10}$$

$\mathbf{z}^{(1)} = \mathbf{Q}^{(1)}\mathbf{q}^{(1)}$. The number m of updates to keep is a hyperparameter. Larger values of m will result in a more accurate approximation of the inverse Hessian, at the cost of increased computational expense. Therefore, the choice of m represents a trade-off between the accuracy of the approximation and the efficiency of the algorithm.

VI. Experiments

In this section, I detail the experiments conducted to solve the Burgers' equation and obtain the uncertain parameters λ_1 and λ_2. As discussed in the 'Approach' section, the ground truth values for λ_1 and λ_2 are 1 and $0.01/\pi$ respectively. The Finite Difference method was employed to generate a training dataset of 25600 samples, each comprising an input pair (x, t) of position and time, and the corresponding output u with added noise. This dataset can be found in the github repository of this project. However, for the purpose of these experiments, a subset of only 2500 samples were randomly selected to demonstrate the effectiveness of Physics Informed Neural Networks (PINNs).

Four experiments were designed for this study, with the aim to conduct a 2-layer comparison: comparing the performance of PINNs against regular Artificial Neural Networks (ANNs), and the effectiveness of the Adam optimizer versus the L-BFGS optimizer in solving this particular problem of PDE discovery.

For all the experiments, a consistent Neural Network architecture was maintained: a 9-layer deep neural network, with 25 neurons per hidden layer. The activation function used between each layer was the hyperbolic tangent function, tanh. The mean square error loss was minimized using different optimization techniques.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 25]              75
            Linear-2                [-1, 25]             650
            Linear-3                [-1, 25]             650
            Linear-4                [-1, 25]             650
            Linear-5                [-1, 25]             650
            Linear-6                [-1, 25]             650
            Linear-7                [-1, 25]             650
            Linear-8                [-1, 25]             650
            Linear-9                 [-1, 1]              26
================================================================
Total params: 4,651
Trainable params: 4,651
Non-trainable params: 0
----------------------------------------------------------------
```

Fig. 1. Model Summary

It's important to note the difference in approach for PINNs and ANNs. PINNs have a custom loss function that considers the PDE in the residual loss component. Here, λ_1 and λ_2 are treated as trainable parameters and are differently incorporated into the loss function. For the PINN experiments, the error percentage for each λ is computed at each epoch and the improvement is plotted in a graph.

For ANNs, however, the approach differs due to the absence of a residual loss component in the loss function. In this case, I first trained the model using the same number of training samples and then used the trained model to predict the output u for all the 25,600 points. With the help of PyTorch's ability to compute the gradient, I also obtained u_t, u_x, and u_{xx}. Then, I fit these points to our PDE $u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0$ using np.linalg.lstsq. The predicted equation is then compared with other experiment results.

A. Experiment 1: PINNs with L-BFGS Optimizer

The approach of using PINNs involves a customized loss function, accounting for the underlying PDE in the form of a residual loss component. This results in λ_1 and λ_2 being declared as trainable parameters and directly incorporated into the loss function.

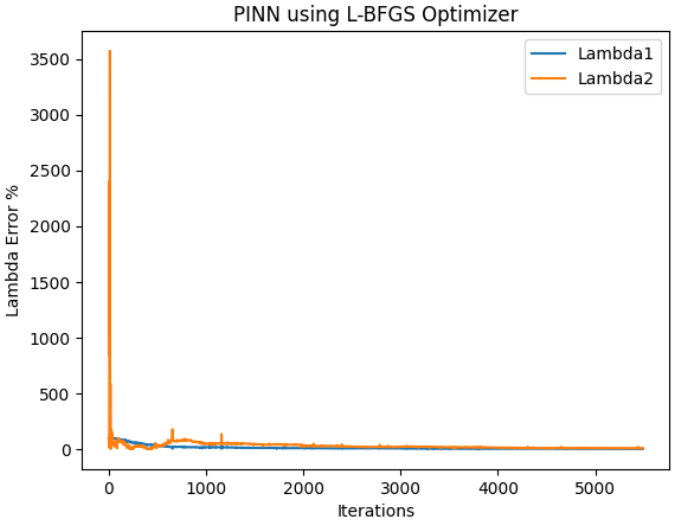

Fig. 2. PINN using L-BFGS

B. Experiment 2: PINNs with Adam Optimizer

This experiment mirrors the previous one, with the only difference being the use of the Adam optimizer instead of L-BFGS.

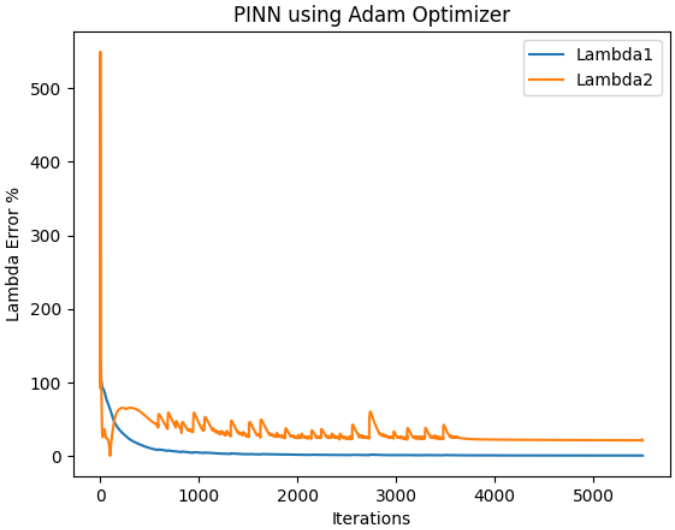

Fig. 3. PINN using Adam

C. Experiment 3: ANNs with L-BFGS Optimizer

The ANNs experiment approach differs fundamentally from the PINNs. Since there is no residual loss component in the

loss function, the method of solving for λ_1 and λ_2 had to be adjusted. Hence in this case there is no prediction of parameters at every epoch, instead there is only one prediction finally.

Parameter	Error Percentage
λ_1	17.82%
λ_2	18.2%

D. Experiment 4: ANNs with Adam Optimizer

This experiment mirrors the previous one, with the only difference being the use of the Adam optimizer instead of L-BFGS.

Parameter	Error Percentage
λ_1	20.23%
λ_2	78.27%

The performance and resulting parameters are compared and discussed in the next section.

VII. Results & Discussion

A. Results

1) Experiment 1: PINNs with L-BFGS Optimizer:

PDE	Form
Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Predicted PDE	$u_t + 0.99005514uu_x - 0.0034076u_{xx} = 0$

2) Experiment 2: PINNS with Adam Optimizer:

PDE	Form
Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Predicted PDE	$u_t + 0.99531293uu_x - 0.003856u_{xx} = 0$

3) Experiment 3: ANNs with L-BFGS Optimizer:

PDE	Form
Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Predicted PDE	$u_t + 0.82174206uu_x - 0.0026034u_{xx} = 0$

4) Experiment 4: ANNs with Adam Optimizer:

PDE	Form
Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Predicted PDE	$u_t + 0.7976258uu_x - 0.00069169u_{xx} = 0$

B. Discussion

From the results, one can observe that the error percentages for Adam optimizer are higher than those for L-BFGS, implying that L-BFGS performed better in these experiments. This is a result of the inherent differences between these two optimization techniques. L-BFGS tends to converge faster and provide more accurate results when dealing with complex problems such as the one I tackled. On the other hand, Adam optimizer, while being simpler and easier to implement, may not always provide the optimal solution.

Another interesting observation from these experiments (also the main goal of this project) is the performance difference between PINNs and ANNs. It can be seen from the results that PINNs outperformed ANNs in terms of parameter prediction accuracy. This can be attributed to the nature of PINNs and their ability to incorporate prior knowledge about the underlying physics of the problem in the form of differential equations into the learning process. This can significantly improve the accuracy and reliability of predictions, especially when dealing with problems that have physical constraints and where data is sparse or noisy.

In the case of ANNs, since they do not incorporate any knowledge about the underlying physics, the prediction results can vary significantly depending on the quality and quantity of the training data and the complexity of the problem.

VIII. Conclusion & Future Work

This project showcased the effectiveness of Physics-Informed Neural Networks (PINNs) in predicting parameters of the Burgers' equation, highlighting their superior performance over traditional Artificial Neural Networks (ANNs). The use of different optimization techniques, namely Adam and L-BFGS, further demonstrated the significant role the optimizer plays in the learning process.

The experimental findings showed that PINNs consistently outperformed ANNs. This is a testament to the efficacy of PINNs in effectively leveraging prior physical knowledge in the learning process, thereby facilitating the prediction of the output and discovering the governing equations. Furthermore, the L-BFGS optimizer provided better performance than Adam, highlighting the significant impact of the choice of optimizer in training these networks.

Therefore, it can be concluded that for problems involving PDEs, PINNs, especially when coupled with the L-BFGS optimizer, can provide more accurate and reliable predictions compared to standard ANNs. This underlines the potential of PINNs as a powerful tool for solving differential equations.

Future work could focus on expanding this study by implementing other optimizers and neural network architectures to augment prediction accuracy. Additionally, exploring the use of these methods with various types of partial differential equations could be an interesting extension of this project. Given the computationally intensive nature of training PINNs and ANNs, future studies might also delve into more efficient algorithms or alternative computational strategies, such as parallel computation or advanced hardware accelerators.

Acknowledgment

I would like to thank Professor Mykel Kochenderfer for introducing the methodologies used in this project through the lectures and the book, "Algorithms for Optimization" [13].

Additionally, since this project was a solo effort, I really appreciate the support and valuable feedback provided by the teaching team of Stanford's "AA 222 / CS 361: Engineering Design Optimization" throughout the course. Their guidance significantly contributed to the completion of this project.

REFERENCES

[1] Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations: Maziar Raissi, Paris Perdikaris, George Em Karniadakis https://arxiv.org/pdf/1711.10566.pdf

[2] Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations: Maziar Raissi, Paris Perdikaris, George Em Karniadakis https://arxiv.org/abs/1711.10566

[3] DeepXDE: A Deep Learning Library for Solving Differential Equations, Lu Lu, Xuhui Meng, Zhiping Mao, George Em Karniadakis https://epubs.siam.org/doi/pdf/10.1137/19M1274067

[4] HyperPINN: Learning parameterized differential equations with physics-informed hypernetworks, Filipe de Avila Belbute-Peres, Yi-fan Chen, Fei Sha https://openreview.net/forum?id=LxUuRDUhRjM

[5] Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, M. Raissi, P. Perdikaris, G. E. Karniadakis, Journal of Computational Physics https://www.sciencedirect.com/science/article/pii/S0021999118307125

[6] PyDEns: a Python Framework for Solving Differential Equations with Neural Networks, Alex Koryagin, er, Roman Khudorozkov, Sergey Tsimfer

[7] NeuroDiffEq: A Python package for solving differential equations with neural networks, Feiyu Chen, David Sondak, Pavlos Protopapas, Marios Mattheakis, Shuheng Liu, Devansh Agarwal, Marco Di Giovanni, Journal of Open Source Software, 2020 https://joss.theoj.org/papers/10.21105/joss.01931

[8] Universal Differential Equations for Scientific Machine Learning, Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, Alan Edelman https://arxiv.org/pdf/2001.04385.pdf

[9] Adversarial uncertainty quantification in physics-informed neural networks, Yibo Yang, Paris Perdikaris, Journal of Computational Physics, 2019 https://www.sciencedirect.com/science/article/pii/S0021999119303584

[10] B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, Liu Yang, Xuhui Meng, George Em Karniadakis, Journal of Computational Physics, 2021 https://www.sciencedirect.com/science/article/pii/S0021999120306872

[11] A physics-aware, probabilistic machine learning framework for coarse-graining high-dimensional systems in the Small Data regime, Constantin Grigo, Phaedon-Stelios Koutsourelakis, Journal of Computational Physics, 2019 https://www.sciencedirect.com/science/article/pii/S0021999119305261

[12] Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs, Siddhartha Mishra, Roberto Molinaro, IMA Journal of Numerical Analysis, 2021. https://academic.oup.com/imajna/article-abstract/42/2/981/6297946?login=true

[13] Algorithms for Optimization, M. J. Kochenderfer and T. A. Wheeler https://algorithmsbook.com/optimization/

[14] https://pytorch.org/docs/stable/index.html

CODE

The code for all the experiments conducted in this project can be found at the following GitHub repository: https://github.com/sumanth107/AA222FinalProject.