Physics Informed Machine Learning for Digital Twins of Metal Additive Manufacturing

Raghav Gnanasambandam

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Industrial and Systems Engineering

Zhenyu Kong, Chair
Xiaowei Yue
Xi Chen
Blake Johnson

April 15, 2024
Blacksburg, Virginia

Keywords: Additive Manufacturing, Deep Gaussian Process, Bayesian Optimization, Physics-informed Neural Networks, Fourier Neural Operators.

Physics Informed Machine Learning for Digital Twins of Metal Additive Manufacturing

Raghav Gnanasambandam

(ABSTRACT)

Metal additive manufacturing (AM) is an emerging technology for producing parts with virtually no constraint on the geometry. AM builds a part by depositing materials in a layer-by-layer fashion. Despite the benefits in several critical applications, quality issues are one of the primary concerns for the widespread adoption of metal AM. Addressing these issues starts with a better understanding of the underlying physics and includes monitoring and controlling the process in a real-world manufacturing environment. Digital Twins (DTs) are virtual representations of physical systems that enable fast and accurate decision-making. DTs rely on Artificial Intelligence (AI) to process complex information from multiple sources in a manufacturing system at multiple levels. This information typically comes from partially known process physics, in-situ sensor data, and ex-situ quality measurements for a metal AM process. Most current AI models cannot handle ill-structured information from metal AM. Thus, this work proposes three novel machine-learning methods for improving the quality of metal AM processes. These methods enable DTs to control quality in several processes, including laser powder bed fusion (LPBF) and additive friction stir deposition (AFSD). The proposed three methods are as follows.

1. Process improvement requires mapping the process parameters with ex-situ quality measurements. These mappings often tend to be non-stationary, with limited experimental data. This work utilizes a novel Deep Gaussian Process-based Bayesian optimization (DGP-SI-BO) method for sequential process design. DGP can model non-stationarity better than a traditional Gaussian Process (GP), but it is challenging for BO. The proposed DGP-SI-BO provides a bagging procedure for acquisition function with a DGP surrogate model inferred via Stochastic Imputation (SI). For a fixed time budget, the proposed method gives 10% better quality for the LPBF process than the widely used BO method while being three times faster than the state-of-the-art method.

2. For metal AM, the process physics information is usually in the form of Partial Differential Equations (PDEs). Though the PDEs, along with in-situ data, can be handled through Physics-informed Neural Networks (PINNs), the activation function in NNs is traditionally not designed to handle multi-scale PDEs. This work proposes a novel activation function Self-scalable tanh (Stan) function for PINNs. The proposed activation function modifies the traditional tanh function. Stan function is smooth, non-saturating, and has a trainable parameter. It can allow an easy flow of gradients and enable systematic scaling of the input-output mapping during training. Apart from solving the heat transfer equations for LPBF and AFSD, this work provides applications in areas including quantum physics and solid and fluid mechanics. Stan function also accelerates notoriously hard and ill-posed inverse discovery of process physics.

3. PDE-based simulations typically need to be much faster for in-situ process control. This work proposes to use a Fourier Neural Operator (FNO) for instantaneous predictions (1000$\times$ speed up) of quality in metal AM. FNO is a data-driven method that

maps the process parameters with a high dimensional quality tensor (like thermal distribution in LPBF). Training the FNO with simulated data from PINN ensures a quick response to alter the course of the manufacturing process. Once trained, a DT can readily deploy the model for real-time process monitoring.

The proposed methods combine complex information to provide reliable machine-learning models and improve understanding of metal AM processes. Though these models can be independent, they complement each other to build DTs and achieve quality assurance in metal AM.

Physics Informed Machine Learning for Digital Twins of Metal Additive Manufacturing

Raghav Gnanasambandam

(GENERAL AUDIENCE ABSTRACT)

Metal 3D printing, technically known as metal additive manufacturing (AM), is an emerging technology for making virtually any physical part with a click of a button. For instance, one of the most common AM processes, Laser Powder Bed Fusion (L-PBF), melts metal powder using a laser to build into any desired shape. Despite the attractiveness, the quality of the built part is often not satisfactory for its intended usage. For example, a metal plate built for a fractured bone may not adhere to the required dimensions. Improving the quality of metal AM parts starts with a better understanding the underlying mechanisms at a fine length scale (size of the powder or even smaller). Collecting data during the process and leveraging the known physics can help adjust the AM process to improve quality. Digital Twins (DTs) are exactly suited for the task, as they combine the process physics and the data obtained from sensors on metal AM machines to inform an AM machine on process settings and adjustments. This work develops three specific methods to utilize the known information from metal AM to improve the quality of the parts built from metal AM machines. These methods combine different types of known information to alter the process setting for metal AM machines that produce high-quality parts.

Dedication

To caffeine, which makes me believe I can do anything.

Acknowledgments

I sincerely thank Prof. Zhenyu (James) Kong, who guided and supported me and my work since the day I submitted my application for a PhD. He and the researchers he created taught me a lot about research, academia, and life. I acknowledge the advisory committee members, Dr. Xiaowei Yue, Dr. Blake Johnson, and Dr. Xi Chen, for giving timely feedback and advice on my research work. Several others played a huge role in navigating through the PhD. In particular, I thank my lab mates and office mates, Dr. Bo Shen and Dr. Jihoon Chung, for their numerous discussions that started at Durham Hall, Room Number 114, and continued virtually. I also thank Benjamin Standfield, who made it easy to focus on research rather than fixing tech issues. I acknowledge the support of other lab members as well, including Dr. Rongxuan Wang, Dr. Andrew Law, Chaoran Dou, and Amirul Islam Saimon, for bringing fresh perspectives and helping me unconditionally. My acknowledgment list will not be complete without including the staff members of Grado Department of Industrial and Systems Engineering at Virginia Tech, who made significant efforts to make my PhD journey easier. Outside work, I thank the constant support of the friends I met at Blacksburg and the Friday evening hangouts. I acknowledge the support of my family for consistently reminding me to take breaks. I also thank the Blacksburg town for being a wonderful place with a reliable bus service.

Contents

List of Figures

List of Tables

List of Abbreviations

BO Bayesian Optimization

DGP Deep Gaussian Process

DT Digital Twin

FNO Fourier Neural Operator

L-PBF Laser Powder Bed Fusion

AM Additive Manufacturing

PINN Physics-informed Neural Networks

Stan Self-scalable hyperbolic tangent

DSVI Doubly Stochastic Variational Inference

EI Expected Improvement

FB Fully Bayesian

MCMC Markov Chain Monte Carlo

N-LAAF Neuronwise Locally Adaptive Activation Function

NN Neural Network

PDE Partial Differential Equation

QoI Quantities of Interest

ROM Reduced-order Model

SI Stochastic Imputation

UCB Upper Confidence Bound

VI Variational Inference

WGP Warped Gaussian Process

Chapter 1

Introduction

1.1 Motivation

Digital Twin (DT) is a virtual representation of any physical entity [131]. A DT typically performs several functions including monitoring, simulation, and planning[137]. Industry 4.0 [26] has enabled continuous sensing with edge computation capabilities that are interconnected. The interconnected nature of the systems allow continuous monitoring and control of variety of physical entities, including manufacturing processes.

Metal Additive Manufacturing (AM) is a manufacturing process to build a part in a layer-by-layer fashion. AM facilitates manufacturing parts with complex geometries that is not possible with traditional manufacturing processes. For instance, Laser Powder Bed Fusion (L-PBF) is a common metal AM process that melts the metal powder to build a part. Despite commonly used, ensuring quality of the products built from L-PBF is difficult. The issue with quality assurance can be primarily attributed to the poor models that represent the process.

Enabling DT for L-PBF requires models that can accurately simulate the process. Modeling an AM process is challenging since the process physics is not well-known [67]. In addition, data-driven modeling techniques are not good enough since the data collection is expensive. Addressing these limitations are necessary for building a DT for L-PBF. This work attempts

to address these limitation in three parts.

1.1.1 Data-Driven Models

Since the L-PBF experiments are expensive, it is necessary to collect the data that is oriented with a goal. This work proposes *DGP-SI-BO* to design expensive experiments that are most useful for the optimization goal. The proposed method uses a Deep Gaussian Process (DGP) [27] as the Bayesian prior. DGP is a composition of multiple traditional Gaussian Processes (GPs) that can model highly non-stationary functions with limited data. Due to the compositional structure, inference in DGP is challenging. Variational Inference (VI) [27, 124] is the most common inference but the poor uncertainty quantification makes it not effective for Bayesian Optimization (BO). In this work, Stochastic Imputation (SI) [95] is used for the posterior inference in DGP. SI inference provides posterior samples that are combined with a novel bagging procedure. The applicability of the proposed algorithm is shown in optimizing the process parameters of L-PBF through simulations.

1.1.2 Physics Informed Models

Another way to minimize the costly experiments is make accurate physics-based simulations that can replicate experiments computationally. Traditional simulations use the first-principles based Partial Differential Equations (PDEs) to simulate the L-PBF process. Apart from the PDEs being an inaccurate representation of the actual process, these simulations use meshing-based numerical methods to solve the PDE system. This work uses Physics-Informed Neural Networks (PINNs) [112] to get more accurate simulations of the L-PBF. PINNs can solve the PDE system mesh-free and can also combine process-data potentially obtained from sensors to get a realistic model. One of the major limitations of PINN is to get multi-scale solutions required for the L-PBF process. This work proposes a novel activation function *Stan* (Self-Scalable Tanh) to efficiently solve PDEs with multi-scale solutions. The

novel activation function is provided with strong theoretical justifications and case studies that extend beyond the L-PBF.

Though the PINNs can provide mesh-free solutions, the PINNs have to be re-trained for individual set of process parameters. The re-training makes the simulations unsuitable for process optimization. In this work, we propose using the Fourier Neural Operator (FNO) [78] as a data-driven model to map the process parameters directly to the simulation output. Results show promising direction in instantaneously predicting the thermal history of a manufactured part with the process parameters.

1.2 Literature Review

Enabling Digitial Twin (DT) for metal Additive Manufacturing (AM) is an actively researched topic [40, 83, 101, 104, 106, 154]. One of the key enablers for DTs is Machine Learning (ML) [24, 65, 126]. ML allows for processing huge amounts of possibly ill-structured data. Traditional ML methods require large amount of data to build an accurate model [46]. Physics-based information can help in decreasing the amount of data required to train a ML model. Likewise, physics-based ML models are used recently in the DT development [81]. This work addresses two main issues in the current literature regarding Physics-based ML.

1. Current Physics-based ML, particularly Physics-informed Neural Networks do not scale for different solutions scales.

2. Current methods do not predict an entire thermal distribution tensor (typically 4-dimensional tensor) from the process parameters (typically less than 10 significant parameters).

Further, uncertainty quantification and subsequent decision-making is not well studied though

necessary for the development of digital twins. This work proposes a Bayesian Optimization method to optimize process parameters of L-PBF under uncertainty. The rest of the chapter reviews specific literature with respect to each of the topics descibed in Chapter 1.

1.2.1 Bayesian Optimization with Deep Gaussian Process

Additive manufacturing (AM) processes (particularly Laser Powder Bed Fusion) are complex with multiple process parameters/control inputs - laser power, scanning speed, powder bed temperature, hatch spacing, etc. These input parameters directly influence the output characteristics like the average dimensional accuracy and mechanical properties like the strength and hardness of the manufactured part, that determine the quality of the process. The relationship between the output characteristics and the input process parameters is usually complex and has no closed-form. Expensive experiments (in terms of cost and time) typically inform these relationships.

Optimal process conditions are crucial for achieving the necessary characteristics of the manufactured part. The optimization problem for finding the best process parameters can be mathematically formulated as optimizing a function f (output property) defined on $x \in \mathcal{X}$, where $\mathcal{X}$ is the process parameters space. The function $f(x)$ is unknown, and thus considered as "black-box". $f(x)$ can represent one or more characteristics of the manufactured part, such as surface quality, distortion in the dimensions of the part, residual stress accumulated, and so on. Without loss of generality, the problem is assumed to be a minimization problem and is formulated mathematically as finding the optimal process parameters,

$$x^* = \arg\min_{x \in \mathcal{X}} f(x). \tag{1.1}$$

Optimizing specific characteristics of the process or part might require repetitive experiments

over various process parameters (inputs) to obtain the right combination [1, 28]. For finding the appropriate optimal process parameters, the number of simulations or experiments must be minimal to reduce costs. Classical Design of Experiments (DoE) [99] provides numerous ways to strategically sample the input space to construct a response surface by optimizing the measurement of input space [51]. These DoE techniques determine the evaluation points before conducting the experiments and are often not very helpful in identifying the optimal process parameters for non-linear response [121].

The model-based experimental design can adaptively sample the input space depending on the response surface and balance exploration and exploitation to obtain a more accurate data model [15, 51, 84]. Among those, Bayesian Optimization (BO) [17, 38, 51, 64, 97, 129] stands out due to its effectiveness in optimizing the response (output) of any expensive "black-box" function. Wide variety of applications use BO, including hyper-parameter tuning in deep learning models [132], designing sensor networks [89], experimental designs in various fields [51], policy search in reinforcement learning [85], preference learning in computer graphics [16], and natural language processing [149].

BO typically consists of two components: (1) a surrogate model of the available data points; and (2) an acquisition function to find the next point to sample. The surrogate model represents the explicit relationship between input and output . The surrogate model should be able to give accurate output predictions with uncertainty quantification to further use an acquisition function. The acquisition function is defined on the posterior of the surrogate model over the support $\mathcal{X}$, and optimizing the function provides the next sampling point. Gaussian Process (GP) regression surrogate model [118] is the widely used surrogate model for BO [38]. Expected Improvement (EI) and Upper Confidence Bound (UCB) are well-known acquisition functions for a GP model [129].

BO does not rely on the structure of the function in terms of the function's derivatives

but relies on the prior assumption of the surrogate model [38]. The prior assumption of a standard GP regression model limits its modeling capabilities to stationary functions. However, the input-output relationship in AM processes are not necessarily stationary. For example, the mechanical properties of a printed metal part, such as strength, ductility, and fatigue resistance, are affected by various process parameters, such as laser power, scanning speed, layer thickness, and powder bed temperature. Increasing the laser power or decreasing the scanning speed may improve the mechanical properties in some regions of the part, but may degrade them in other regions. Similarly, changing the layer thickness or powder bed temperature may have different effects depending on the geometry of the part and the material being printed. This motivates the investigation of a better surrogate model for modeling the non-stationarity in AM processes.

[56] and [105] used direct formulations to extend the GPs to non-stationary functions whereas [48] used a Locally approximate GP (LaGP) model for the same. These techniques either involve high parameterization or use some information on the structure of the function, making these methods not quite useful for BO with limited data. [133] used a warping approach to use a standard GP for BO, but the approach is not as expressive as a Deep Gaussian Process (DGP) [54].

Previous studies suggest the DGP model [27], inspired by the Deep Learning [46], for modeling the general class of functions [95, 128]. A "multi-layered" DGP is similar to a neural network consisting of layers of perceptrons [46]. A DGP uses multiple GPs combined to project the input space into alternate spaces defined by the GP layers to conveniently model the non-stationarity. Several studies point to the better expressiveness of a DGP over a GP in a small data regime [27, 31, 95, 124, 128]. Also, the DGPs have been proven to be superior to the GPs in several surrogate modeling tasks [54, 113, 128].

Nevertheless, using the DGP for BO is quite challenging in two aspects: (1) the inference

method; and (2) the acquisition function. A fully Bayesian (FB) Markov Chain Monte-Carlo (MCMC)-based inference [128] conceivably addressed the former, since the state-of-the-art method, Doubly Stochastic Variational Inference (DSVI) [124] has poor uncertainty quantification. For the acquisition function, [50] provided a way to use the posterior samples of FB inference by averaging a traditional acquisition function (EI) over the samples.

However, the MCMC-based procedure is notably slow in handling sizeable data with several process parameters. Stochastic Imputation (SI)-based inference [95], also a sampling-based method, promises a faster inference while preserving the benefits of the FB inference. This work extends [50]'s averaging of acquisition function over the components by a bootstrap aggregation (bagging) for SI-based inference. Specifically, this work proposes an algorithm *DGP-SI-BO* to facilitate the use of DGPs for BO using SI-based inference. The case studies suggest that the proposed algorithm is better suited for BO than the FB inference based BO [50]. The main contributions of this work are as follows.

- Propose a procedure to use the SI-based inference [95] for BO using a DGP surrogate model. Specifically, an acquisition function, defined as a Sample Average Approximation (SAA) [130] over several components of the posterior distribution [50], can be improved using bootstrap aggregation (bagging) [5]. Given a dataset, bagging is helpful in reducing the variance of the estimated next evaluation point for BO.
- Present an algorithm *DGP-SI-BO* which uses the proposed methodology. Multiple analytical test functions and a case study in AM simulation testify the usefulness of the presented algorithm.

1.2.2 Multi-scale Solutions for PINNs

Deep Learning (DL) has revolutionized many facets of our modern society, such as self-

driving vehicles, recommendation systems, and web search tools. Apart from the end-user applications, DL is also accelerating solutions to problems in engineering and science, such as fluid dynamics[70], transportation systems[150], geotechnical engineering[151], manufacturing[111], biomedical engineering[36], biology[146], and physics[136]. Despite the successes when analyzing complex physical systems, much of the DL literature is purely data-driven, ignoring a vast amount of prior knowledge. This knowledge is typically in the form of differential equations from the principled physical laws.

From the seminal work on solving differential equations with a Neural Network (NN)[72] and its major revival as Physics-informed Neural Network (PINN)[112], it is striking that a simple NN can inherently solve any general differential equation (with boundary conditions). Since then, a wide variety of applications (e.g.,[12, 18, 32, 88, 116, 119, 122, 125]) used PINNs to model processes with little to no data. PINN is typically a multi-layered fully-connected neural network[66] designed for the mapping between the input spatio-temporal location and the output, which is the solution to be learned. The loss function embeds the differential equations together with the fitting loss of available data (like initial/boundary conditions or experimental data), such that learning the weights (and biases) of the NN directly implies learning the solution of the differential equation.

Traditional methods to numerically solve differential equations, like the Finite Element Methods (FEM)[29], create a mesh to interpret the solution approximately at discrete points in the spatio-temporal domain. On the other hand, PINNs can solve the differential equations mesh-free, provide a pseudo closed-form solution, and can also combine the data from actual experiments to generate a more accurate model[20]. Further, PINNs can solve the inverse problem of discovering the parameters of the differential equations provided the observed experimental data[66, 112]. The inverse problem is unsolvable with the traditional numerical methods and not well explored by the standard DL methods.

Despite the extraordinary advantages, the success of PINN is limited to a handful of representative problems, and broader adoption is still far-fetched due to a wide range of issues. The unconventional loss function with multiple competing objectives explains the poorly understood loss landscape and substandard training [143, 145]. One line of work addresses the convergence of the loss function through adaptive hyper-parameters (e.g.,[60, 61, 63, 91, 143]). Wang et al. [143] proposed a learning-rate annealing procedure to adaptively weigh the different parts of the loss function to overcome the difference in the convergence between these parts, whereas McClenny et al.[91] provided a way to adaptively weigh the sampled points that constitute the loss function. Both of these works use the tanh activation function and rely on the statistics of components of the loss function to improve the convergence. Jagtap et al. (refer to [60, 61, 63]) used adaptive activation functions to improve the convergence for any general NN problem. Another line of work utilizes domain-decomposition-based approaches to avoid the *spectral-bias*[1] of NN [100] and obtain well-scaled solutions (e.g.,[59, 62, 100]).

This work examines the loss function of PINNs to analyze the poor training from the point of view of the activation function, particularly the tanh[75] function. The widely-used tanh function

1. has conflicting requirements on its first derivative,

2. can lead to poor gradients for the physics-informed loss function, and

3. can fail to efficiently solve problems with multi-scale solutions.

These issues motivate the design of a new activation function. This work proposes a novel activation function, Self-scalable tanh (Stan), for extending PINNs to solve a variety of

[1]Spectral-bias is the term referred to the NN model learning the lower frequencies faster than the higher frequencies.

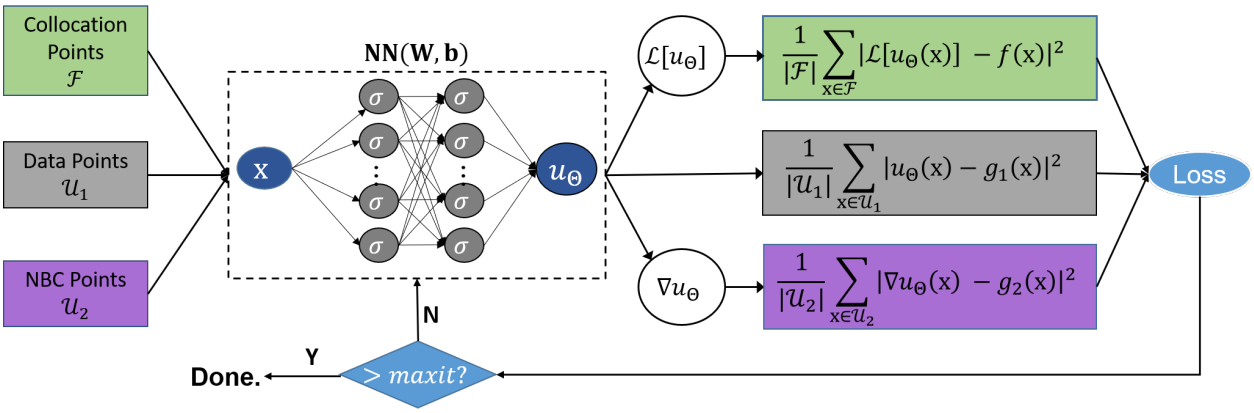


Figure 1.1: Schematic of PINN for a general differential equation system. The training of the PINN starts with the sets of points $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$ which are subsets of Ω,$\partial\Omega_1$, and $\partial\Omega_2$, respectively. The points in set $\mathcal{F}$ form the first part of the loss function, the points in set $\mathcal{U}_1$ form the second part, and the points in set $\mathcal{U}_2$ form the third part, as determined by the NN at current training step. The loss is back-propagated to train the parameters of the NN.

differential equation systems at different scales. The Stan function has a tanh term and an additional self-scaling term with a trainable parameter similar to the Swish function[114]. To summarize, the contributions of this paper are as follows:

- Derive the first and the second derivative terms of any differential equation for a general fully-connected NN model. The derivation provides insights into some failure modes in training PINN.

- Derive the gradients of the loss function for training the PINN. The gradient calculations provide unique requirements concerning the higher-order derivatives of the activation function.

- Propose a new Self-scalable tanh (Stan) activation function, which enables learning outputs (solution) with values of potentially higher orders of magnitude.

- Empirically show superior training and better solutions of differential equation systems with the proposed activation function in both forward problems (to obtain the

approximate solutions) and inverse problems (to identify the parameters involved in the governing equation), compared to the state-of-the-art activation functions.

Related Work in Activation Functions

Designing activation functions for NNs has been an active research area[6]. The activation function and its gradient play an important role in the training process, influencing the gradients of the loss function for the optimization of parameters (weights and biases)[52]. A poorly designed activation function causes loss of input information during forward propagation and exponential vanishing/exploding of gradients during back-propagation, leading to poor convergence. There is no single activation function that works well for all the problems in DL[115].

For PINN, the tanh[75] and its adaptive modification[60, 61] have been the most successful activation functions. The popular Rectified Linear Units (ReLU)[103] are not suitable for PINN[96]. Hennigh et al.[55] and Al-Safwan et al.[3] had some successful cases with the Swish activation function. More recently, Jagtap et al.[63] proposed the Rowdy activation function, which adds sinusoidal fluctuations with trainable parameters to the tanh activation function. Nevertheless, these works do not consider the unique loss function formulation of PINN.

1.2.3 Fast Thermal Simulations

Quality issues in a metal AM process mainly arise from poorly designed process parameters [76]. Physics-based simulations [47] indirectly inform these process parameters. These numerical simulations predict the quality metrics or Quantities of Interest (QoI) given a process parameter setting with a Partial Differential Equation (PDE)-based model [47]. The

PDE-based models are fundamental physical laws that typically predict temperature and displacement at a macroscale that define the QoI by quantifying the heat flow and the mechanical forces during the AM process [34]. Simulations allow experimenting with different process parameter settings, solving a PDE-based model for each setting, and optimizing the QoI. However, these PDE-based simulations are time-consuming [4], even for a single design, making design optimization expensive.

One way to reduce the time consumed by repeated experimentation is to use a surrogate model-based sequential experimental design for one or more predefined quality goals [44]. However, this approach is inefficient since changing the design goal requires new experiments for a new surrogate model. Additionally, this approach does not allow an understanding of the exact influence of process parameters, preventing possible expert decisions on redesign or post-processing of the concerned part. Thus, the entire PDE solution, i.e., temperature or displacement as a spatial-temporal function, is generally desirable with lesser computational time than the simulation software.

Reduced-order models (ROMs) [8], traditionally derived through techniques such as proper orthogonal decomposition [22] or proper generalized decomposition [25], are less time-consuming than the high-fidelity simulations [35, 41]. With the advent of data-driven methods in recent years, the traditional ROMs are being replaced by Deep Learning (DL)–based ROMs [39, 98, 108]. These DL-based ROMs have been vastly adopted to accelerate simulations due to their flexibility in learning complex non-linear relationships without explicit mathematical formulation.

In the context of AM, DL has been used for various tasks with data from experiments and simulations alike [92, 123]. Firstly, most applications of DL in AM focus on predicting a categorical variable (classification-type tasks), such as detecting and identifying defects in the built part. In addition, the works that focus on predicting continuous variables (regression-

type tasks), such as melt pool dimensions [2], focus on one or, at most, a few quality metrics of interest. Predicting few quality metrics is reasonable for experimental-data-based DL models due to the limitations of metrology equipment, and the natural world is much more complex than the physics-based models.

In the case of DL-based ROMs, since the QoI are derived fundamentally from the solutions of PDEs, it is desirable to build unified models that predict the entire solution (for example, temperature as a function of spatial-temporal location at a given process parameter setting). This line of work has not been explored well since the notable DL models like ResNet [53], UNet [120], and Vision Transformer [30] require large volumes of data to train. Physics-informed neural networks (PINNs), a relatively new class of DL, predict the required entire solution by directly solving PDEs, like simulations, and do not need any data upfront. For instance, with the PDE-based heat transfer model, [152] and [82] predict temperature as a spatial-temporal function. Nevertheless, PINNs are time-consuming and require re-training for new process parameters [152], making them as disadvantageous as other numerical simulators for design problems.

To address these challenges, this work proposes to use Fourier Neural Operators (FNO) [78] as a DL-based ROM to simulate AM processes. The proposed method has three main advantages.

1. **Predict entire PDE-solutions:** Instead of focusing on a few quality metrics, FNO predicts the solution of PDEs by learning the mapping from process parameters to spatial-temporal solution function.

2. **Instantaneous Predictions:** Once trained, the FNO can predict the entire spatial-temporal function at any given process parameter in a fraction of a second. Thus, a single model can be used in various contexts with preferred quality metrics or design

requirements.

3. **Fewer Data to train:** Compared to other DL methods, the FNO trains with a small amount of data. In the context of AM, this allows for a realistic number of data-generating simulations to build a high-accuracy model.

FNO is part of a class of methods that learn operators [69] that map between function spaces, whereas traditional DL methods learn maps between discretized spaces (like vectors, matrices, or tensors). FNO has previously been used in AM [23, 138, 148]. [148] used FNO to predict the maximum bead volume and the maximum melt pool temperature in Directed Energy Deposition (DED). The proposed method calculates QoI with Multiphysics Object Oriented Simulation Environment (MOOSE) [153] before training instead of predicting the entire solution of PDE. [138] used FNO to predict the final state of droplet coalescence from an initial state in Liquid Metal Jet AM. [23] utilized FNO to capture local temperature evolution in DED, by training with temperature data at successive timesteps as input and output. These works did not include time-varying output, which is essential for holistically analyzing the influence of process parameters.

The main contribution of this work is to provide an FNO-based framework to essentially capture the entire high-dimensional solution of PDE for varying process parameters. Provided process parameters (typically 1-3 parameters used in the case studies), the proposed method predicts the entire thermal distribution as a function of spatial-temporal location. Training the proposed data-driven models takes only a few hundred simulations, which is far fewer and more realistic to utilize in metal AM.

1.3 Dissertation Organization

The rest of the dissertation is organized as follows. Chapter 2 provides the details on DGP-SI-BO algorithm and Chapter 3 provides the details on Stan activation function. Chapter 4 gives the study on using FNO for fast thermal simulations. Chapter 5 provides conclusions and future work.

Chapter 2

Bayesian Optimization with Deep Gaussian Process

The chapter organization is as follows. Section 2.1 gives the theoretical background of various constituents, which lead to the proposed Bayesian Optimization with Stochastic Imputation-bsed inference of Deep Gaussian Process (*DGP-SI-BO*) algorithm. Section 2.2 describes the proposed methodology in detail before demonstrating the use cases in Section 2.3. Section 2.5 summarizes the work and discusses some future directions.

2.1 Bayesian Optimization Theory

Let $f : \mathcal{X} \in \mathbb{R}^d \rightarrow \mathbb{R}$ be the expensive "black-box" function to be optimized. Assuming we have n observations of the function f at $\mathbf{X} = (x_1, x_2, ..., x_n)$ where $x_i \in \mathcal{X} \; \forall \; i \in \{1, 2, 3, ..., n\}$, let $\mathbf{y} = (y_1, y_2, ..., y_n)$ denote $(f(x_1), f(x_2), ..., f(x_n))$. The problem is defined to find x^* as in Eq. (1.1) in a minimum number of function evaluations. In BO, a surrogate model is built based on these n observations $(\mathbf{X}, \mathbf{y})$, and a new point x_{n+1} is computed by optimizing the acquisition function defined on the model for the next evaluation of the "black-box" function f. The surrogate model predicts the functional value of f with uncertainty estimates at any $x \in \mathcal{X}$. The acquisition function uses these predictions to get x_{n+1} to reach the optimal functional value in a minimum number of evaluations.

2.1.1 Surrogate Model

Gaussian Process

Let $\mu_0(\cdot)$ and $\Sigma_0(\cdot,\cdot)$ be the prior mean and prior covariance of the given data, respectively. Then the prior distribution of f is given as

$$P(f(\cdot)) \sim \mathcal{GP}(\mu_0(\cdot), \Sigma_0(\cdot,\cdot)). \tag{2.1}$$

For the GP prior in Eq. (2.1), finite observations of f form a joint Gaussian distribution. The mean function $\mu_0(.)$ is typically considered as $\mathbf{0}$, and the covariance $\Sigma_0(\cdot,\cdot)$ is typically of the squared-exponential kernel form parameterized by τ, λ, and g as in Eq. (2.2).

$$k(x,\acute{x}) = \tau^2(\exp(-\frac{|x-\acute{x}|^2}{\lambda}) + g) \;\forall\; \{x,\acute{x}\} \in \mathcal{X} \tag{2.2}$$

The parameters (τ, λ, and g) of the kernel function are usually estimated by Maximum Likelihood (ML) estimation. It turns out that the posterior predictive distribution of the functional value at any new observation $f(x)$ at x given $(\mathbf{X},\mathbf{y})$ is also Gaussian [118], with a closed-form mean and variance, as in Eq. (2.3).

$$\begin{aligned} \Pi(f(x)|x,\mathbf{X},f(\mathbf{X})) &\sim \mathcal{N}(\mu_n(x),\sigma_n^2(x)), \\ \mu_n(x) &= \mu_0(x) + \Sigma_0(x,\mathbf{X})\Sigma_0^{-1}(\mathbf{X},\mathbf{X})(f(\mathbf{X}) - \mu_0(\mathbf{X})), \\ \sigma_n^2(x) &= \Sigma_0(\mathbf{X},\mathbf{X}) - \Sigma_0(x,\mathbf{X})\Sigma_0^{-1}(\mathbf{X},\mathbf{X})\Sigma_0(\mathbf{X},x). \end{aligned} \tag{2.3}$$

Limitations of GP Regression

Though there are significant advantages in using a GP regression model, the structure of the covariance function, which is usually a function of $(x - x')$ (for example, the kernel in

Eq. (2.2)), limits the GP's ability to model functions that exhibit non-stationarity. [56] proposed a class of non-stationary kernels based on spatially-varying kernel functions, and extended by [105] later to give a closed-form non-stationary covariance function. Eq. (2.4) provides the spatially-varying squared exponential function [105].

$$k(x, \acute{x}) = \tau^2 \left|\Sigma_x\right|^{\frac{1}{4}} \left|\Sigma_{\acute{x}}\right|^{\frac{1}{4}} \left|\frac{\Sigma_x + \Sigma_{\acute{x}}}{2}\right|^{-\frac{1}{2}} \exp(-Q(x, \acute{x})) \ \forall \ \{x, \acute{x}\} \in \mathcal{X} \tag{2.4}$$

where

$$Q(x, \acute{x}) = (x - \acute{x})^T \left(\frac{\Sigma_x + \Sigma_{\acute{x}}}{2}\right)^{-1} (x - \acute{x}),$$

Σ_x and $\Sigma_{\acute{x}}$ are covariance matrix of the Gaussian kernel centered at x and $\acute{x}$, respectively. The spatially-varying covariance matrix $\Sigma(\cdot)$ is typically highly parameterized [105], and therefore [105] use the domain knowledge of the data to parameterize the covariance matrix. Thus, the non-stationary kernels are unsuitable for a general "black-box" optimization.

Another approach, to address the stationary covariance structure, is to define locally stationary GPs that would give globally non-stationary behavior [37, 48, 49, 117]. For instance, [48] used local data subsets when the size of the training data is high. Since the data are small in the BO task, local GPs may not have sufficient data to estimate the parameters [54].

Yet another approach is to warp the input space [133, 141]. [133] used the *Beta CDF* to warp the input x to $w(x)$, so that any stationary kernel can be used in the warped space (an example using squared-exponential function is provided in Eq. (2.5)). The approach is limited by the choice of transformation function w and does not capture non-stationarity as effectively as a DGP [54].

$$k(x, \acute{x}) = \tau^2(\exp(-\frac{|w(x) - w(\acute{x})|^2}{\lambda}) + g) \ \forall \ \{x, \acute{x}\} \in \mathcal{X} \tag{2.5}$$

Deep Gaussian Process

To motivate the modeling with DGP, consider the GP modeling in Eq. (2.1). Assume that the response data $\mathbf{y}$ is not available, and instead, the problem is to find a low dimensional representation of data $\mathbf{X}$. Specifically, let W_1 (of a given dimension) be the latent (unobserved) low dimensional representation of $\mathbf{X}$ and the problem is to identify Θ_1^* that projects $\mathbf{X}$ to the given low dimension, where Θ_1 denote the parameters of the GP (for the prior in Eq. (2.1), the parameters are τ, λ, and g in Eq. (2.2)). To find Θ_1^*, define a prior distribution $p(W_1)$ and marginalize it, and maximize the likelihood as

$$\begin{aligned} P(\mathbf{X}|\Theta_1) &= \int P(\mathbf{X}|W_1, \Theta_1) p(W_1) dW_1 \\ \Theta_1^* &= \arg\max_{\Theta} P(\mathbf{X}|\Theta_1). \end{aligned} \tag{2.6}$$

The integral in Eq. (2.6) is intractable even for prior $p(W_1)$ taken as Gaussian. Thus, [73] suggested that a Gaussian prior is defined on Θ_1, and the likelihood of W_1 is maximized. It turns out that the posterior $P(\mathbf{X}|W_1)$ is tractable, and the low dimensional representation W_1 can be obtained in closed-form [73]. This is the Gaussian Process-Latent Variable model (GP-LVM) [73], which is a particular probabilistic interpretation of the Principal Component Analysis (PCA) [139].

[74] further showed that the W_1 could be warped into W_2, W_3, and so on, creating a hierarchy of GP-LVMs. [27] proposed to marginalize the latent variables in the hierarchy ($W_l \; \forall \; l \in \{1, 2, 3, ..., L\}$, where L is the number of latent variables) through *variational inference* (VI) [140]. The variational marginalizing of the latent variables by forcing independence between them was the earliest inference method for a DGP. A DGP prior can

produce highly non-stationary mappings due to the warping of input in each layer [31]. For surrogate modeling with a DGP, the output of the final GP in the hierarchy (also called leaf node) is observed, i.e., $\mathbf{y}$ (Figure 2.1). The prior distribution for the model in Figure 2.1 is given as

$$\begin{aligned} P(f(\cdot)|W_1) &\sim \mathcal{GP}(\mathbf{0}, \Sigma_{W_1}) \\ P(W_1) &\sim \mathcal{GP}(\mathbf{0}, \Sigma(\cdot,\cdot)), \end{aligned} \tag{2.7}$$

where the mean functions are usually taken as zero functions, and Σ_{W_1} represent the prior covariance matrix. The prior distribution of intermediate variables (W_1) is taken to be a (multivariate) Gaussian distribution with covariance Σ.

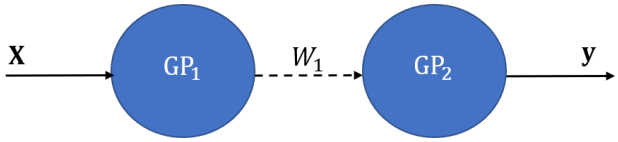


Figure 2.1: A simple 2-layered DGP is represented in this figure. Each circle represents a node which is a GP. Each $\text{GP}_m \ \forall \ m \in \{1, 2\}$ is characterized by its corresponding parameters $\Theta_m \ \forall \ m \in \{1, 2\}$. $\mathbf{X}$ and $\mathbf{y}$ are observed variables, and W_1 is a latent variable.

Though the success of the DGPs are primarily due to the advancement in VI, VI is not well suited for BO due to poor estimates of variance. [128] used a fully-Bayesian (FB) approach to infer the posteriors of parameters $\Theta_m \ \forall \ m \in \{1, 2, 3, ..., L, L{+}1\}$, latent variables $W_l \ \forall \ l \in \{1, 2, 3, ..., L\}$, and $f(x)$ by utilizing an MCMC-based posterior sampling through a hybrid of Gibbs-Metropolis and Elliptical Slice Sampling (ESS) [102]. The FB inference is free of tuning parameters and provides robust variance estimates.

More recently, [95] proposed the Deep Gaussian Process Stochastic Imputation (DGP-SI). The idea of DGP-SI is to convert a DGP into multiple Linked GPs. A Linked GP (LGP) [94] is similar to a DGP (in Figure 2.1), but the intermediate layer outputs $W_l \ \forall \ l \in \{1, 2, 3, ..., L\}$ are observed rather than inferred. Thus, for the DGP inference, the intermediate layers are

imputed multiple times, like in a missing data problem, to get multiple LGPs [95]. The SI method is promising in terms of speed and accuracy for BO with DGP.

Though DGPs are shown to be better at modeling non-stationary functions, particularly in small data regime [27, 95, 128], there is less research on the usage of DGPs for BO, mainly because of the inference methods. [54] used the DSVI method [124] for the DGP surrogate model and approximated the posterior predictive distribution as Gaussian. The approximation is nonviable, especially considering that DSVI [124] itself is an approximation method with poor variance estimates [95, 128]. [50] and [127] utilized the FB approach for BO but the approach is practically slow. This motivates us to choose the SI-based inference method, which has better estimates for variance than DSVI and computationally faster than FB inference. The details of the DGP-SI and the subsequent use of it in the BO are discussed in Section 2.2.

2.1.2 Acquisition Function

For sampling the next evaluation point, an acquisition function $a(x)$ must be defined over the support $\mathcal{X}$ with the current surrogate model at every step. The acquisition function is then optimized over the support to obtain the next evaluation point x_{n+1},

$$x_{n+1} = \arg\max_{x \in \mathcal{X}} a(x). \tag{2.8}$$

The acquisition function is less expensive to evaluate than the "black-box" function f. There are a handful of widely used acquisition functions for the surrogate methods. The most prominent acquisition functions in the literature are Expected Improvement (EI) [132] and Upper Confidence Bound (GP-UCB) [134].

The Gaussian posterior, with $\mu(x)$ and $\sigma(x)$ representing the mean and the standard deviation, respectively, gives a closed-form expression (Eq. (2.9)) for the EI acquisition function. f^{best} denotes the best (minimum for a minimization problem) value of "black-box" function obtained so far. ϕ and Φ denote the *pdf* and *cdf* of the standard Gaussian distribution, respectively.

$$EI(x) = (\mu(x) - f^{best})\Phi\left(\frac{\mu(x) - f^{best}}{\sigma(x)}\right) + \sigma(x)\phi\left(\frac{\mu(x) - f^{best}}{\sigma(x)}\right). \tag{2.9}$$

On the other hand, GP-UCB is more intuitive acquisition function with theoretically guaranteed convergence for a Gaussian posterior distribution [134]. For BO, using the posterior predicted mean function $\mu(x)$ alone would define a pure exploitation strategy and using the predicted standard deviation function $\sigma(x)$ alone would define a pure exploration strategy; combining both with a parameter β has proven to be very useful for GP-based BO [134]. For a maximization problem, the GP-UCB (or simply UCB) function in Eq. (2.10) is maximized to obtain the next evaluation point.

$$UCB(x) = \mu(x) + \beta^{1/2} \times \sigma(x). \tag{2.10}$$

Since the DGP surrogate model makes the posterior $P(f(x)|x, \mathbf{X}, \mathbf{y})$ non-tractable, these acquisition functions cannot be directly used for BO but need approximations. [54] successfully used the EI with the Gaussian approximation of DSVI. [50] and [127] used the EI with the FB inference by averaging the function over the posterior samples. While both EI and UCB acquisition functions can be applied in SI-based BO, this work is the first to specifically examine the use of UCB for this purpose.

2.2 Proposed DGP-SI-BO

The outline of the methodology is shown in Figure 2.2. The algorithm starts from an initial dataset modeled by the DGP surrogate as described in Section 2.2.1. The SI-based inference in the surrogate model leads to the novel bagging approach developed for the acquisition function as described in Section 2.2.2. Optimizing the acquisition function gives the next evaluation point x_{n+1}. The value $f(x_{n+1})$ is identified by evaluating the "black-box" function and is added to the dataset. The process of adding a datum continues till a stopping criterion is encountered.

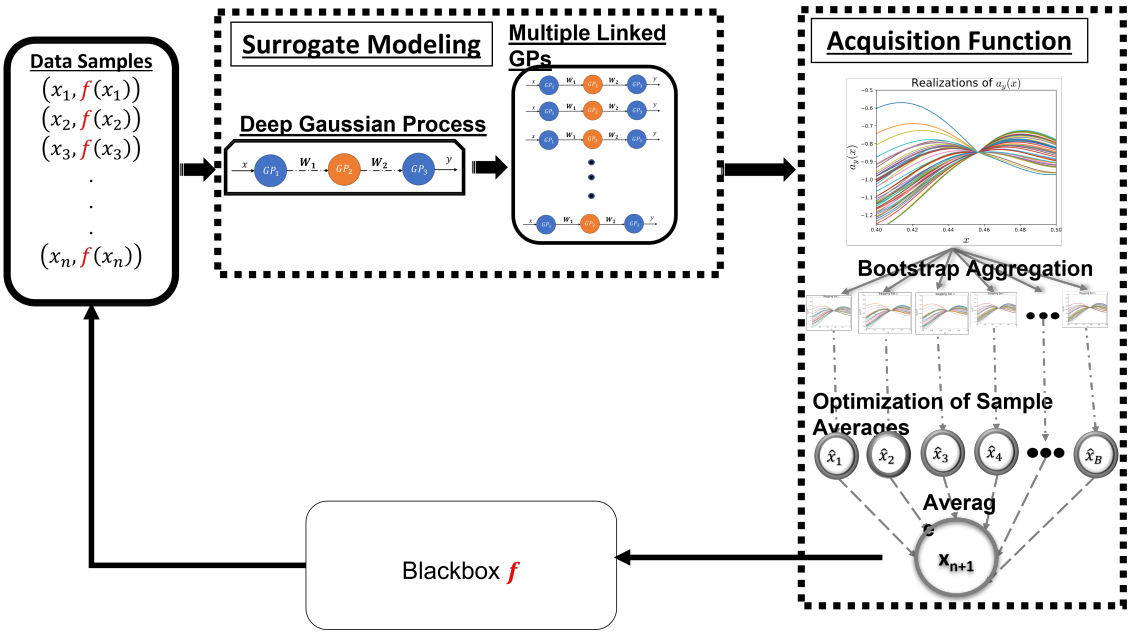


Figure 2.2: The DGP-UCB algorithm showing the main components - (1) DGP-SI that convert the DGP into multiple linked GPs for inference of the surrogate model; (2) The acquisition function that uses the inference structure to provide the next evaluation point for the "black-box" function.

2.2.1 DGP-SI Inference

Let $\mathbf{W} = \bigcup_{l=1}^{L} W_l$ represent all the combined intermediate latent variables of the DGP model, and let $\mathbf{\Theta} = \bigcup_{m=1}^{L+1} \Theta_m$ denote all the parameters of all the GPs in the DGP. Unlike the FB inference [128], DGP-SI has a training procedure to infer the optimal parameters $\mathbf{\Theta}^*$ and has a distinct prediction by imputation of intermediate variables $\mathbf{W}$ for $\mathbf{\Theta}^*$ [95].

Training

The SI inference consists of iterating two steps [95] - (1) Imputation of $\mathbf{W}_{(s)}$ through MCMC-based procedure and (2) Maximizing the likelihood of individual GPs to obtain $\mathbf{\Theta}_{(s)}$ at any training step $s, s \in \mathbb{N}$. The SI inference is different from the FB approach of [128] as the FB approach uses a full posterior sampling of both $\mathbf{\Theta}$ and $\mathbf{W}$.

Prediction

For prediction, the latent variables are imputed to get T realizations $\mathbf{W}_{(t)}\ \forall\ t \in \{1, 2, 3, ...T\}$ of the intermediate layers at the given $\mathbf{\Theta}^*$. A realization gives the complete pseudo data $\{\mathbf{X}, \mathbf{W}_{(t)}, \mathbf{y}\}$ corresponding to an LGP. Each of these realizations forms an LGP with posterior predictive distribution $\Pi(f(x)|x, \mathbf{W}_{(t)}, \mathbf{X}, \mathbf{y})$. These posterior predictive distributions can be approximated ($\hat{\Pi}(f(x)|x, \mathbf{W}_{(t)}, \mathbf{X}, \mathbf{y})$) as Gaussian distributions for the prominently used kernels [71, 94]. Thus, with the SI, the posterior predictive distribution $(f(x)|x, \mathbf{X}, \mathbf{y})$ for the DGP is given as [95]

$$\begin{aligned} (f(x)|x, \mathbf{X}, \mathbf{y}) &= \int (f(x)|x, \mathbf{W}, \mathbf{X}, \mathbf{y})\Pi(\mathbf{W}|\mathbf{X}, \mathbf{y})d\mathbf{W} \\ \implies (f(x)|x, \mathbf{X}, \mathbf{y}) &\approx \frac{1}{T}\sum_{t=1}^{T} \hat{\Pi}(f(x)|x, \mathbf{W}_{(t)}, \mathbf{X}, \mathbf{y}). \end{aligned} \tag{2.11}$$

With the Gaussian approximation of LGP, the distribution $(f(x)|x, \mathbf{X}, \mathbf{y})$ is an equally weighted mixture of Gaussian. For BO, this non-Gaussian posterior is unsuitable for most of the standard acquisition functions, and thus some approximations are required, as described next.

2.2.2 Acquisition function for DGP-SI

There are several ways to compute the acquisition function for the BO with DGP-SI.

- Using the Gaussian approximation, similar to the work of [54], it is possible to obtain the acquisition function (as in Eq. (2.9) and Eq. (2.10)) of the DGP model directly by the mean $(\mu_y(x))$ and variance $(\sigma_y^2(x))$ estimates. But the Gaussian assumption is too simplistic and is not well founded.

- The DGP can be approximated to an LGP by using the expected value of intermediate variables $\mathbf{W}$. In this case, the estimated mean $(\hat{\mu}_y^{\mathbb{E}[\mathbf{W}]}(x))$ and the estimated variance $(\hat{\sigma_y^{\mathbb{E}[\mathbf{W}]}}(x))$ are different[1] from the actual mean $(\mu_y(x))$ and variance $(\sigma_y^2(x))$ estimates. Note that the realizations of $\mathbf{W}$ for the SI-inference depend only on the data $\{\mathbf{X}, \mathbf{y}\}$ and the inferred $\mathbf{\Theta}^*$ (and not on x) [95]. For instance, Eq. (2.12) provides the UCB function for the new estimates. The expectation $\mathbb{E}[\mathbf{W}]$ has to be approximated as an average of the realizations $\mathbf{W}_{(t)}\ \forall\ t \in \{1, 2, 3, ..., T\}$.

$$\tilde{a}(x) = \hat{\mu}_y^{\mathbb{E}[\mathbf{W}]}(x) + \beta^{1/2}(\hat{\sigma_y^{\mathbb{E}[\mathbf{W}]}}(x))^2, \tag{2.12}$$

  This approach is again simplistic as it ignores the distribution of $\mathbf{W}$.

[1] These estimates can be directly calculated from Eq.(3) and Eq.(4) of [95]. The value of intermediate variables should be taken as $\mathbb{E}[\mathbf{W}]$.

- Considering each realization $\mathbf{W}_{(t)}$ of the intermediate variables, it is feasible to define a function $a_{y_{(t)}}(x)$ denoting the acquisition function at that realization. For instance, Eq. (2.13) denote the UCB function defined on an LGP. Note that imputing $\mathbf{W}$ for T times implies that T number of LGPs are generated.

$$a_{y_{(t)}}(x) = \mu_{y_{(t)}}(x) + \beta^{1/2} \times \sigma_{y_{(t)}}(x) \; \forall \; t = \{1, 2, 3, ..., T\}. \tag{2.13}$$

  It is practical to optimize the functions $a_{y_{(t)}}$ individually ($\forall \; t \in \{1, 2, 3, ..., T\}$), and obtain the optimum $\hat{x}_t \; \forall \; t \in \{1, 2, 3, ..., T\}$ corresponding to each LGP (and thus, corresponding to each realization $\mathbf{W}_{(t)}$). Perhaps, the average of $\hat{x}_t \; \forall \; t \in \{1, 2, 3, ..., T\}$ can provide the required x_{n+1}. The limitation of using this approach is the increasing number of optimizations with increasing T.

- For the mixture distribution, [50] resort to Monte Carlo integration, i.e., simply averaging the acquisition function (in this case, EI) of all the components ($a_{y_{(t)}} \; \forall \; t \in \{1, 2, 3, ..., T\}$) of the mixture (also known as SAA). Since the distribution of the estimated x_{n+1} is unknown, we extend [50]'s averaging strategy to propose a bootstrap aggregation (bagging) [5] procedure to reduce the variance in estimation of x_{n+1}.

Using the Monte Carlo integration [50] with bagging, the DGP-SI is empirically shown to optimize the functions faster than the FB inference-based BO. Though bagging can be used with either type of inference, the effect of bagging is more pronounced in SI inference-based BO since the number of prediction samples (T) is practically quite small. [95] recommend a value of $T = 50$, which is far too less than the number of available samples from the FB inference (typically a few thousands). Increasing T can significantly affect the computation time in practice, and unnecessary for the inference [95].

Bagging for Improved Estimation

Note that $\mathbf{W}_{(t)}$ is a realization of the random variable (intermediate latent variable) $\mathbf{W}$, and $a_{y_{(t)}}(x)$ is a realization of a random function (or random process) $a_y(x)$. The aim is to optimize the expectation of the function $a_y(x)$ to get the next evaluation point of the BO algorithm as

$$x_{n+1} = \arg\max_{x \in \mathcal{X}} a(x) = \arg\max_{x \in \mathcal{X}} \mathbb{E}[a_y(x)]. \tag{2.14}$$

The problem in Eq. (2.14) is a stochastic optimization problem where we observe only the function realizations of the random function $a_y(x)$. Specifically, by Monte Carlo integration (also known as Sample Average Approximation in the field of Stochastic Optimization [130]), the function $\mathbb{E}[a_y(x)]$ can be approximated by the average of sampled realizations as given by

$$\mathbb{E}[a_y(x)] \approx \frac{1}{T} \sum_{t=1}^{T} a_{y_{(t)}}(x). \tag{2.15}$$

Substituting Eq. (2.15) in Eq. (2.14), we get

$$x_{n+1} = \arg\max_{x \in \mathcal{X}} \frac{1}{T} \sum_{t=1}^{T} a_{y_{(t)}}(x). \tag{2.16}$$

It is known that the expectation converges to the actual function as $T \to \infty$. The distribution of the estimated x_{n+1} is certainly unknown, and a simple averaging scheme might give a poor estimation of mean of x_{n+1}. For DGP-SI, instead of employing the costly MCMC-based sampling, a simple bagging procedure can improve the computation of x_{n+1} with the minimal number of samples. Thus, out of the T realizations of function $a_y(x)$, R realizations are sampled (with replacement) for B times to form the sets of realizations $A_1, A_2, ..., A_B$.

Algorithm 1 DGP-SI-BO Algorithm

Require: $\mathbf{X}, f(\mathbf{X}), T, maxeval, B, R.$
  $i \leftarrow 1;$
  $b \leftarrow 1;$
  $\mathbf{y} \leftarrow f(\mathbf{X}).$
  while $i \leq maxeval$ **do**
    $t \leftarrow 1;$
    $\mathbf{\Theta}^* \leftarrow \text{DGP}(\mathbf{X}, \mathbf{y});$
    while $t \leq T$ **do**
      Obtain a realization of $\mathbf{W}_{(t)}$ by imputation at $\mathbf{\Theta}^*$;
      Calculate $\mu_{y_{(t)}}(x)$ and $\sigma_{y_{(t)}}(x)$ by LGP Gaussian approximation;
      $a_{y_{(t)}}(x) \leftarrow \mu_{y_{(t)}}(x) + \beta_i^{1/2}\sigma_{y_{(t)}}(x);$
    end while
    while $b \leq B$ **do**
      $A_b \leftarrow bootstrap(\{a_{y_{(t)}}(x)\ \forall t = 1, 2, 3, ..., T\})$ with$|A_b| = R;$
      $\hat{x}_b \leftarrow \arg\max_{x \in \mathcal{X}} \frac{1}{R}\sum_{a_{y_b} \in A_b} a_{y_b}(x);$
    end while
    $x_{n+1} \leftarrow \frac{1}{B}\sum_{b=1}^{B} \hat{x}_b;$
    $\mathbf{X} \leftarrow append(\mathbf{X}, x_{n+1});$
    $\mathbf{y} \leftarrow append(\mathbf{y}, f(x_{n+1}));$
    $i \leftarrow i + 1;$
  end while
  $index \leftarrow \arg\max_{k \in \{1,2,...,maxeval\}} y_k.$
  return $x_{index}.$

For each set, the optimum $\hat{x}_b\ \forall\ b = 1, 2, ..., B$ is identified by SAA as

$$\hat{x}_b = \arg\max_{x \in \mathcal{X}} \frac{1}{R} \sum_{a_{y_b} \in A_b} a_{y_b}(x)\ \forall\ b = 1, 2, ..., B. \tag{2.17}$$

Once the B optimization problems are solved, the overall optimum is calculated as the mean of the optimum of each set [5], i.e.,

$$x_{n+1} = \frac{1}{B} \sum_{b=1}^{B} \hat{x}_b. \tag{2.18}$$

A common practice is to choose $R = T$, and the value of B is usually much greater than

T [5]. Eqs. (2.17) and (2.18) are used in calculating the next evaluation point for the original "black-box" function f. The data point $(x_{n+1}, f(x_{n+1}))$ is added to the dataset $(\mathbf{X}, \mathbf{y})$, and the DGP model is rebuilt to repeat the procedure till the stopping criterion is reached. Algorithm 1 provides the complete proposed algorithm.

2.3 Analytical Tests

Since bagging is crucial for defining the acquisition function in the Algorithm 1, Section 2.3.1 shows a 1D test case to visualize and validate the effect of bagging. Further, the algorithm is tested on multiple hard-to-optimize analytical test functions and Section 2.3.2 provides the details of the same. The discussion in Section 2.4 shows the usefulness of the algorithm in a case study in metal AM simulation to identify the optimal process parameters. All the codes are implemented on an Intel(R) Core(TM) i9-9940X CPU @ 3.30GHz with 128.0 GB RAM.

The *dgpsi* Python package [95] is used for implementing the proposed DGP-SI-BO. In Section 2.3.2 and Section 2.4, the proposed algorithm is compared with the benchmark methods including standard GP based BO (GP-BO) [38], GP based BO with input-warping (WGP-BO) [133], DGP-BO using DSVI (DGP-DSVI-BO) [54], DGP-BO using MCMC inference (DGP-MCMC-BO) [50], and random optimization.

For implementing the conventional GP-based BO, Sci-kit learning's GP model [109] is used to find the posterior mean and variance. The acquisition for the GP model is directly calculated from the predictions as in Eqs. (2.9) and (2.10). For the WGP-BO, BoTorch [9] is utilized for implementing the *Single Task* GP by warping the input, and this work follows the EI acquisition function [133]. The DGP-DSVI-BO [54] is implemented with GPFlux package [33] for the DGP modeling with DSVI. In the work of [54], the posterior is directly

assumed as a Gaussian distribution to utilize EI acquisition function. Thus, the obtained mean and variance from DSVI of the DGP are used directly to calculate the *pdf* and *cdf* of the Gaussian distribution to get the value of the EI function (as in Eq. (2.9)). Squared exponential kernels are used for all the cases across all the algorithms.

For the UCB-based algorithms, the value of β is taken as $\beta = 0.2 \times \log(2 \times i \times d)$ as suggested by [134], where i denotes the number of points in the dataset $(\mathbf{X}, \mathbf{y})$, and d is the dimension of x. Unlike GP-based BO algorithms, the acquisition function do not have a closed-form expression, and sampling from the posterior predictive distribution is computationally costly. To overcome the cost, [128] used a fixed candidate dataset to search for the optimal of acquisition criterion whereas [50] used a triangulation approach to choose the candidate points at every step. In the current work, either of the approaches are used depending on the context (detailed later), and a simple exhaustive search algorithm (direct implementations are available in Python and R) to search within the candidates. Irrespective of the inference method, a 2-layered DGP surrogate with the dimension of intermediate variable same as the input dimension is utilized for all the cases.

2.3.1 1D Test Function

To demonstrate the effectiveness of bagging, a 1D non-stationary function is defined as

$$f(x) = \begin{cases} \sin(\pi x + \pi) + x + \cos(2\pi x + \pi), \text{ if } 0 \leq x < 0.3 \text{ or } 0.6 < x \leq 1 \\ -0.9 + 0.1 \sin(50\pi x), \text{ if } x \geq 0.3 \text{ and } x \leq 0.6. \end{cases} \tag{2.19}$$

The 1D function is assumed to be the "black-box" function. The BO task is to minimize the given function in minimum number of function evaluations. A DGP model with a one-dimensional intermediate variable $\mathbf{W}$ is used as the surrogate model. The results are reported

on UCB-based acquisition function[2].

The number of training iterations is set to 500. Following the work of [95], the number of realizations of the latent variables, which consequently lead to the number of components in the mixture Gaussian, is set to 50. Appendix A.2 provides a simple illustration of bagging procedure.

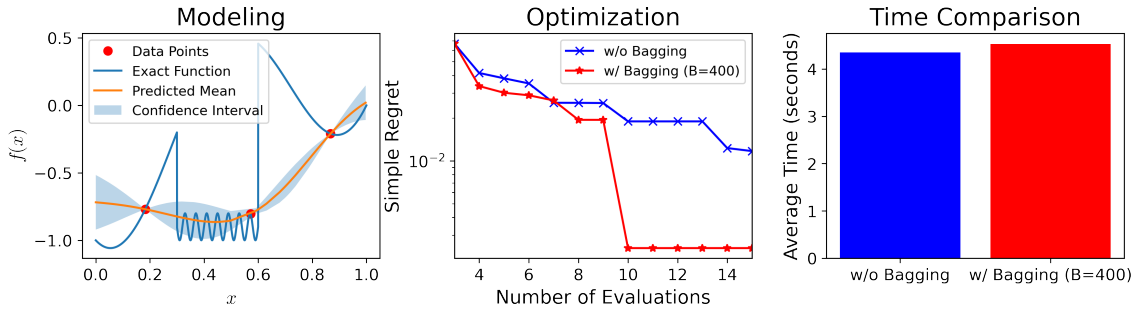


Figure 2.3: (Left) Visualization of the modeling with DGP-SI with the three sampled points; (Center) Comparison of the Simple Regret values obtained over the next twelve evaluations repeated five times with a UCB-based acquisition function; (Right) Comparison of time taken per step for the two approaches (with and without bagging). Bagging does not increase the computational time significantly.

At every BO step, the acquisition function is minimized on a uniformly chosen 100 random points in the support. For each realization $\mathbf{W}_{(t)}$ of $\mathbf{W}$, there is a realization $a_{y_{(t)}}(x)$ of $a_y(x)$. Among the 50 realizations, R (which is again taken as 50) number of realizations are chosen, with replacement after every selection, to form one bagging set. The bagging is repeated for B times (taken as 400) to get 400 sets of 50 "pseudo"-realizations. The bagging procedure brings about 400 optimal values $\hat{x}_b \; \forall \; b = 1, 2, 3, ..., 400$. These $\hat{x}_b \; \forall \; b = 1, 2, 3, ..., 400$ are averaged to obtain the next evaluation point at every step.

The mean $\mu_y(x)$ and uncertainty estimates ($\pm 1.96 \times \sigma_y(x)$) from modeling with the DGP-SI with 3 points in the support $\mathcal{X}$ (sampled by Latin Hypercube Sampling (LHS)) are shown in

[2]For a minimization problem, the second term in Eq. (2.10) must be subtracted from the $\mu(x)$ term to give the "Lower Confidence Bound" (LCB). Though LCB is used for the minimization problems, the rest of the chapter still uses "UCB" for consistency.

Figure 2.3(a). The initial estimates are poor due to inadequate data. The surrogate model is further used for the BO task for twelve more steps. Figure 2.3(b) shows the significance of bagging in optimization of the 1D function(averaged over 5 repetitions). As observed from Figure 2.3(b), the bagging procedure can get closer to the actual optimum value in fewer steps due to better utilization of the minimal mixture components. Figure 2.3(c) shows that the bagging procedure can lead to increase in computation time but the increase is very small in comparison with the time taken for building the surrogate model with DGP-SI.

2.3.2 Literature Test Functions

The proposed DGP-SI-BO algorithm is tested on several difficult-to-optimize analytical test functions from literature [135]. Similar to the 1D test function (in Section 2.3.1, the analytical test functions are considered "black-boxes" for testing the algorithm. In this work, apart from the 1D test function, *Six-Hump camel, Three-Hump camel, Hartmann3D, Hartmann4D,* and *Michalewicz 10D* functions [135] are implemented to assess the proposed algorithm. The function equations are provided in Appendix A.3.1.

After the initial set of evaluations through $5 \times d$ number of LHS samples, several additional points are added by BO to compare the algorithms. For all the functions except the *Michalewicz 10D*, the support $\mathcal{X}$ is discretized with triangulation [50] for choosing the next evaluation point. The optimal point (x_{n+1}) is then used for evaluating the "black-box" function, and the point is added to the already evaluated dataset of points. For *Michalewicz 10D*, due to the computational burden of high-dimensional function, a set of 1000 points are chosen beforehand, to search for the optimal within the dataset. Choosing a fixed number candidate points is a common practice for high-dimensional functions [128]. Further, it is observed that using a pre-determined candidate dataset can lead to poor convergence with

utilizing UCB-based acquisition functions. Thus, for *Michalewicz 10D*, EI-based acquisition function is used for for GP-BO and DGP-SI-BO whereas for all other functions, UCB-based acquisition functions give satisfactory performance.

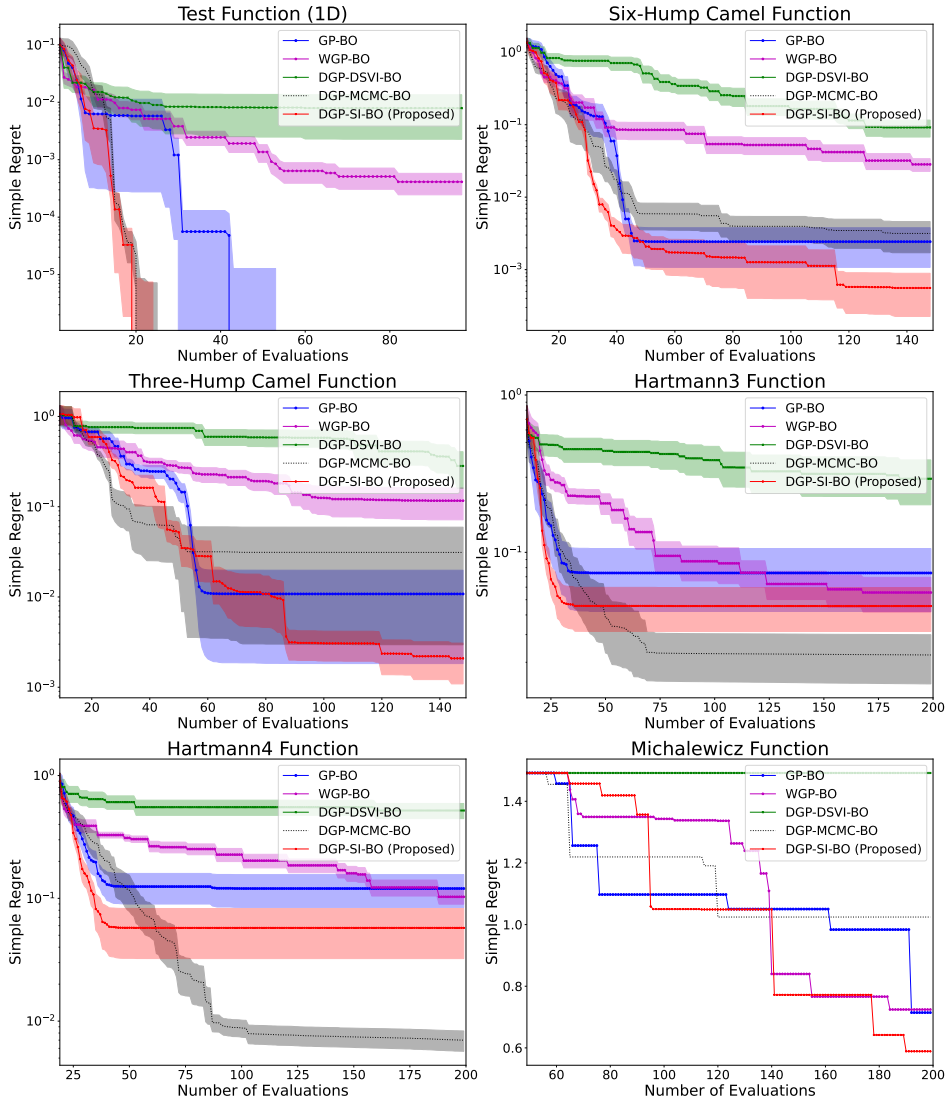


Figure 2.4: Averaged Simple Regret plots for the analytical test functions for all the considered algorithms. For *Six-Hump camel, Three-Hump camel,* and *Michalewicz*, the proposed algorithm converge to the lowest value whereas for *Hartmann3D* and *Hartmann4D*, DGP-MCMC-BO converges to a better value.

The details of all the algorithms including parameter choices are provided in Appendix A.3.3.

For the DGP-SI-BO, the acquisition function is optimized for each bootstrap aggregated set

$A_b \; \forall \; b \in \{1, 2, ..., B\}$ (B values are in Appendix A.3.3) to get $\hat{x}_b \; \forall \; b \in \{1, 2, ..., B\}$, on the discrete points. Except *Michalewicz 10D*, x_{n+1} for DGP-SI-BO is calculated as in Eq. (2.18). For *Michalewicz 10D*, among the 1000 chosen points, the point that minimizes the Euclidean distance of the average in Eq. (2.18) is taken as x_{n+1} .

For all the considered algorithms, the Simple Regret (SR) [86] values as a function of number of "black-box" evaluations are plotted in Figure 2.4. The values at the end of stipulated number of steps obtained from all the algorithms are tabulated in Table 2.1. Overall, the proposed method is better at optimizing the analytical test functions in terms of SR. Though the DGP-MCMC-BO's [50] SR values are better than the proposed method in *Hartmann3* and *Hartmann4* functions, the values of the DGP-MCMC-BO are worse than the GP-based BO for *Six-Hump camel, Three-Hump Camel,* and *Michalewicz 10D* functions.

Table 2.1: Final averaged Simple Regret (SR) values with their standard errors at the end of optimizing the analytical test functions. The red bold numbers indicate the smallest SR value and the black bold numbers indicate the second smallest value for the given function. The standard error values of Michalewicz function are skipped since a fixed candidate set is used for optimization.

Algorithm / **Function**	GP-BO [38]	WGP-BO [133]	DGP-DSVI-BO [54]	DGP-MCMC-BO [50]	DGP-SI-BO (Proposed)
Test Function (1D)	**0.0000 (3.80e-7)**	**0.0004 (1.67e-4)**	0.0079 (5.60e-3)	**0.0000 (3.43e-7)**	**0.0000 (5.59e-7)**
Six-Hump Camel	**0.0024 (0.0014)**	0.0283 (0.0056)	0.1418 (0.0271)	0.0032 (0.0014)	**0.0006 (0.0003)**
Three -Hump Camel	**0.0108 (0.0090)**	0.1166 (0.0450)	0.0224 (0.0115)	0.0312 (0.0282)	**0.0021 (0.0010)**
Hartmann3	0.0739 (0.0319)	0.0555 (0.0138)	0.2937 (0.0932)	**0.0223 (0.0077)**	**0.0456 (0.0142)**
Hartmann4	0.1202 (0.0359)	0.1029 (0.0133)	0.5193 (0.0731)	**0.0070 (0.0002)**	**0.0575 (0.0252)**
Michalewicz (10D)	**0.7145 (–)**	0.7245 (–)	1.4919 (–)	1.0245 (–)	**0.5888 (–)**

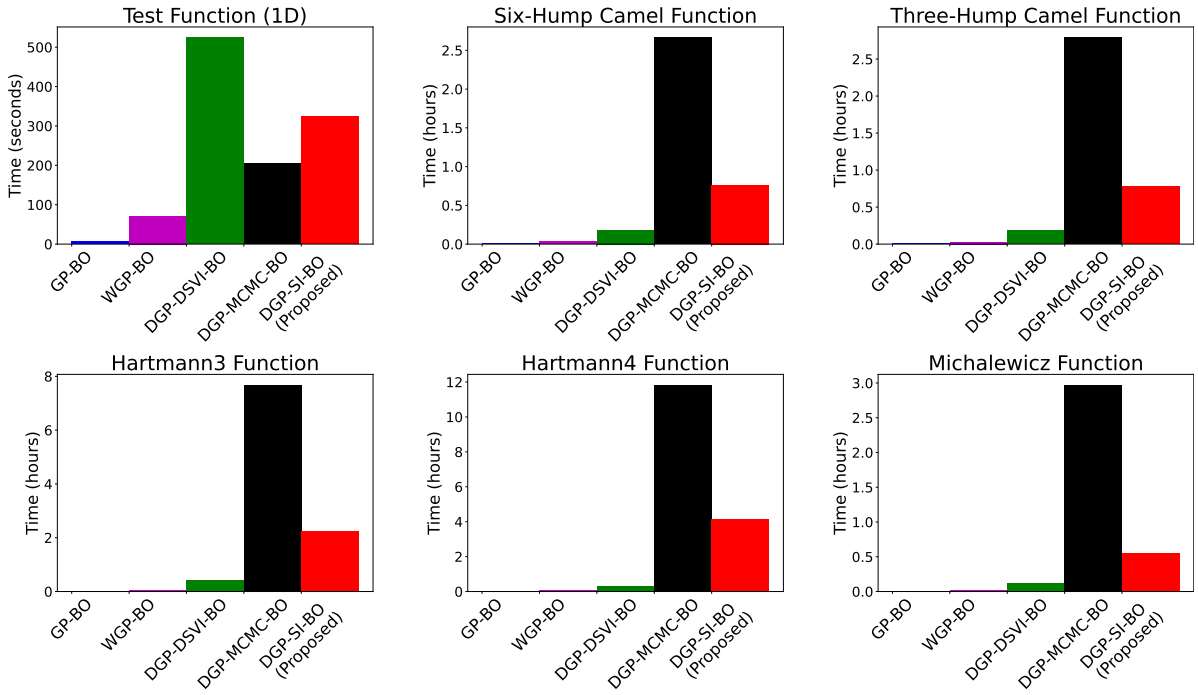


Figure 2.5: Comparison of the average computation time on the analytical functions for a single BO task. Note that the evaluation of "black-box" function is instantaneous for all the cases. The 1D test function does not follow the trend as the function typically reaches the satisfactory optimal value much earlier than the stipulated number of steps.

The average computation times for a single evaluation step are noted and tabulated in Table 2.2 for all the analytical functions. Though the proposed DGP-SI-BO provide competing results in terms of SR, the computation times of the proposed method and DGP-MCMC-BO [50] are still much higher than the other comparison methods. For instance, the proposed DGP-SI-BO is approximately 5 times slower than the DGP-DSVI-BO and approximately 10^2 times slower than the GP-BO, as seen in Table 2.2. The slowness of DGP-SI-BO and DGP-MCMC-BO can be traded-off with optimization accuracy since the BO is usually used for "black-box" experiments whose evaluation time and cost might be much higher than what it takes for the algorithm to obtain the next evaluation point. Table 2.2 also shows that the DGP-MCMC-BO is approximately 3 times slower than the proposed algorithm. Conceivably, the same argument on slowness cannot be applied to DGP-MCMC-BO in comparison

with DGP-SI-BO since the accuracy of DGP-SI based BO is better than the FB DGP-MCMC based BO. It is also observed that the B number of optimizations for the proposed algorithm does not affect the computation time as much as increasing the number of emulation steps of DGP-SI.

For the sampling-based surrogates (DGP-SI and DGP-MCMC), the computation times differ drastically with the size of the data and dimensionality of the input (and the intermediate variable). For instance, the computation times of DGP-SI and DGP-MCMC are plotted against the size of the data for *Six-hump camel* function in Figure 2.6. Though both the sampling-based algorithms slow down with the increase in size of the data, the rate of slow down for FB inference with DGP-MCMC is much higher than the DGP-SI. Figure 2.5 shows the cumulative time as bar charts for all the analytical test functions per entire run of BO. The ratio of cumulative times of DGP-MCMC-BO and DGP-SI-BO is higher than the ratio of inference times due to the slowdown with the increase in size of the data.

Table 2.2: Approximate computation times (in seconds) comparing the algorithms for evaluating a single step including building the surrogate, optimizing the acquisition function, and evaluating the "black-box" function (negligible because of availability of closed-form expression) for analytical functions. The values of the proposed method are shown in bold.

	GP-BO [38]	WGP-BO [133]	DGP-DSVI-BO [54]	DGP-MCMC-BO [50]	DGP-SI-BO (Proposed)
6-Hump Camel	0.1642	1.5874	26.9895	76.7331	**24.4778**
3-Hump Camel	0.2507	1.1377	26.9681	80.2826	**24.9886**
Hartmann3	0.2153	1.6542	10.1296	157.4055	**48.2676**
Hartmann4	0.2558	2.1577	7.3927	250.5618	**89.3343**
Michalewicz (10D)	1.6841	0.4403	8.0330	148.7114	**34.2952**

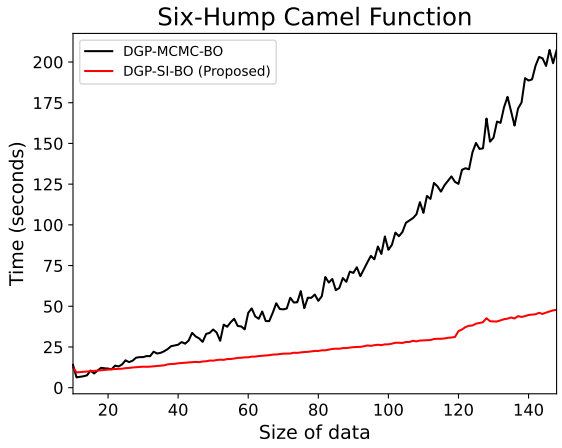
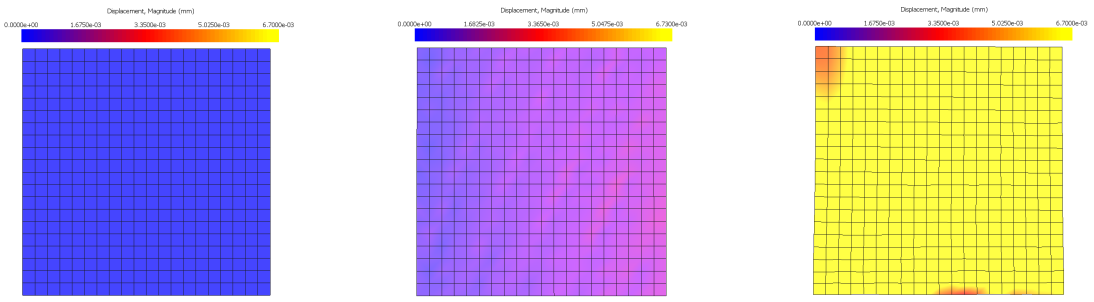


Figure 2.6: The average computation time of DGP-SI-BO and DGP-MCMC-EI (over 10 repetitions) for an example analytical function as the size of data increases. The DGP-MCMC-BO slows down more significantly than the proposed DGP-SI-BO.

2.4 Process Parameter Optimization in AM

The DGP-SI-BO algorithm is applied to a case study with application in metal AM. To build a part with AM, a number of process parameters are needed to be decided beforehand. Inappropriate process parameters can lead to poor part quality, loss of material, labor, and time, and increased cost. When we need a perfect part while manufacturing the part, the process parameters need to be appropriate in terms of qualities of interest. The process parameters depend on the material used, the geometry of the manufactured part, surrounding conditions, etc., to get the required quality. The process parameters are required to be tuned either based on the expert's domain knowledge or simple trial and error to figure out the combinations that work best. For the Laser Powder Bed Fusion (LPBF) process, a laser melts the metal powder to guide the manufacturing process. The main influencing parameters of the part quality are laser power, laser scanning speed, and hatch spacing. To overcome the impractical repeated trails to close in on the appropriate process parameters, BO provides a way to sequentially sample the support to get to the optimal input process parameters.

Simulations mimic the actual experiment and can be considered a way to determine the process parameters. Simulation is a mathematical model of the actual process and can help determine some critical quality parameters. Finite Element Analysis (FEA) [47] based simulations are popular in modeling the metal AM process. FEA simulations use a meshing-based numerical approximation of the mathematical model to mimic the process. Though FEA simulations are helpful, they are still costly in terms of operation cost and time. Autodesk Netfabb Local Simulation [7] is the state of the art FEA package used for thermal and mechanical simulations for metal AM. For this case study, a simple cuboid of shape $2mm \times 2mm \times 0.2mm$ is chosen as the part to be studied. $0.2mm$ is also determined as the thickness of the layer. The average quality measures are studied for a single-layer print. The material is set to Ti-6Al-4V with properties at standard room temperature conditions. The process parameters range from $50W$ to $416.5W$ for laser power, from $122.65mm/s$ to $1421.69mm/s$ for scanning speed, and from $0.0597mm$ to $0.2043mm$ for hatch spacing.



(a) 0.0021mm (b) 0.0032mm (c) 0.0137mm

Figure 2.7: Visualization of displacement values as obtained from the Autodesk Netfabb Simulation software. The corresponding average Displacement values are mentioned.

For this work, two output properties are considered to be optimized simultaneously. The first is the average displacement (distortion) of the manufactured part, sufficient time after the process. With the dimensions of the considered part, and the time taken to finish the process, a duration of $60s$ is set as a time when the part would have sufficiently cooled

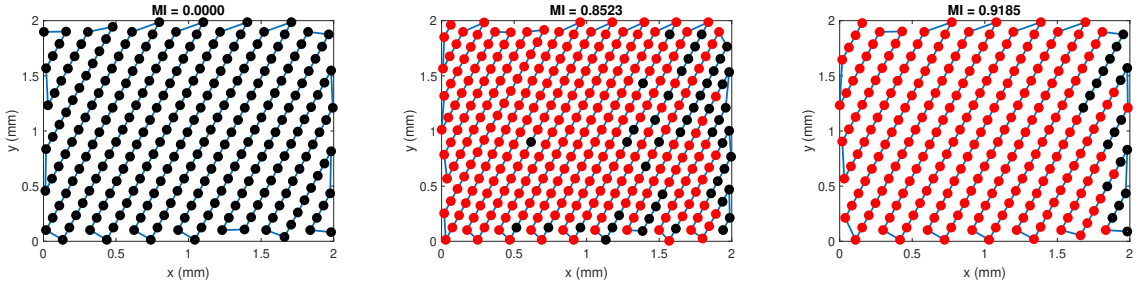


Figure 2.8: Visualization of the melt indicator on a single layer for three different values. The blue line indicates the laser path, and the dots mark the time steps that are recorded by the software. Red dots indicate the time steps that had a temperature greater than $1600K$, and the black ones indicate the steps that had a temperature less than $1600K$.

to observe the distortions. The distortion is caused by a complex combination of melting, solidification, and reheating. The distortion should be minimized, and often the part design comes with tolerance specifications of the dimensions that are to be met to qualify the part. The software output visualization for the displacement values from the top view is shown in Figure 2.7. The color map indicates the level of displacement at that position from the intended geometrical location. Figure 2.7a shows a relatively low displacement, whereas Figure 2.7c shows a relatively highly distorted part.

The second output property considered is an average indicator of appropriate melting by the laser. The melting quality determines various characteristics of the manufactured part like microstructure, which in turn influences the physical properties. For better quality, it is desired that all the powder in the laser path are melted. Therefore, the melt indicator (MI) is calculated as the fraction of time steps that resulted in melting the powder as in Eq. (2.20). The simulation software has a set minimum time step for which the software can indicate the temperature at the position of the laser. $1600K$ is considered appropriate as a preheat temperature for the titanium alloy (Ti-6Al-4V) for melting the powder. A visualization of

the MI values is provided in Figure 2.8.

$$MI = \frac{\#\text{ of time steps} > 1600K}{\text{Total }\#\text{ of time steps}}. \tag{2.20}$$

The first property needs to be minimized, and the second property needs to be maximized. Since the laser can fail to melt the powder at all times, the second property value can be zero and undesired in the denominator. Thus, the two output properties are combined by simply dividing the MI value by AD value. The combined objective needs to be maximized, and thus the problem is mathematically written as

$$\{LP^*, SS^*, HS^*\} = \underset{\{LP,SS,HS\}}{\arg\max} \frac{MI}{AD}, \tag{2.21}$$

to find the optimal laser power, scanning speed, and hatch spacing. For BO, the same procedure and the same benchmark methods used for the analytical test functions are followed. Initially, $5 \times dimension$, i.e., 15 LHS samples, are chosen from the support of the process parameters. These samples are checked for modeling capabilities by LOOCV, and the correlation plot is provided in Appendix A.4.1. To reduce the effort of repeated manual simulations, a set of 1000 candidate simulations are used for making a dataset. To compare the proposed algorithm with benchmark methods, one of the data point is chosen from the 1000 candidates at every BO step. The parameter choices for the algorithms are provided in Appendix A.4.2.

Table 2.3: The optimal objective values obtained and the average time taken (per entire run) for the various algorithms on the AM simulation-based evaluations. Note that the time for the AM simulation is not included in the provided time calculations.

	GP-BO [38]	WGP-BO [133]	DGP-DSVI-BO [54]	DGP-MCMC-BO [50]	**DGP-SI-BO (Proposed)**
Optimum Value (mm^{-1})	162.2890	109.6758	154.8992	177.8386	**178.9485**
Time (seconds)	0.4728	1.1042	6.1381	67.8220	**36.4713**

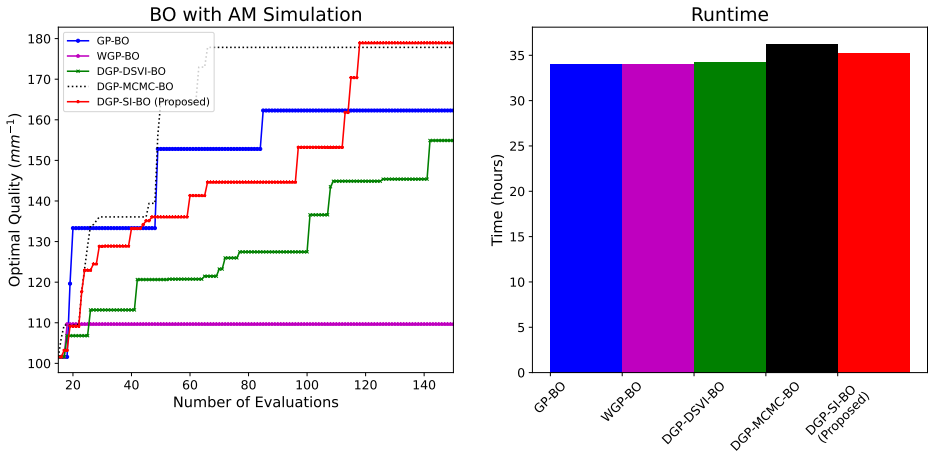


Figure 2.9: Comparison of various algorithms on the AM simulation data as an average optimal value of quality as function of number of evaluations (left), and in terms of computation time per run considering each simulation takes 15 minutes (right). The exact simulation time is unknown, and involves human-in-the-loop to get the values.

The average optimum values from all the considered algorithms are tabulated in Table 2.3. From the table, it can be clearly seen that the DGP-SI-BO reaches a better value of quality than the other benchmark algorithms. The performance in terms of the obtained quality is also seen from the plot in Figure 2.9. Figure 2.9 shows the time comparisons with bar charts of all the algorithms counting the AM simulation time as well. Even for a simple cubic part chosen for this study, the AM simulation takes most of the computational resources making the computation time of the algorithm less significant for "black-box" optimization.

2.5 Summary

A bagging procedure for BO with a DGP surrogate model is proposed. The suggested algorithm takes advantage of the LGP structure-based DGP-SI inference, providing better estimates of the sequential samples in BO by bagging. The algorithm is shown empirically powerful in optimizing non-stationary functions compared to other benchmark methods,

exhibited by the optimization of analytical test functions. The algorithm is also applied to a metal Additive Manufacturing simulation-based case study of a non-stationary function. The algorithm was able to identify the best set of input parameters consistently for optimizing a defined quality objective based on melt indicator and average displacement. The poor performances of the benchmark methods can be mainly attributed to the small data regime and the simplification of posterior distribution through inference and acquisition function.

Bagging can reduce the variance of estimated x_{n+1}, irrespective of the inference method, and one of the future directions is to apply the bagging procedure on other inference methods, particularly the FB inference. Though the proposed method is well reasoned for the distinctive performance, the method needs more theory in the future to back the results and to quantify the convergence of the algorithm. Extending the algorithm to optimize multiple objectives instead of combining multiple objectives into a single one is also a suggested future direction. The extension should be straightforward as the LGP structure allows for multiple outputs directly. Apart from number of posterior realization samples, higher dimensional latent variables and deeper structures are required in higher dimensions, reducing the computational speed. The computational aspect of the proposed algorithm needs further investigation as well.

Chapter 3

Self-scalable Tanh (Stan) Activation Function for Physics-informed Neural Networks

The chapter organization is as follows. Section 3.1 introduces the proposed methodology with the Self-scalable tanh activation function with its motivation and properties. Section 3.2 provides the results on NN approximation of discontinuous function, PINN solutions of one-dimensional differential equations, and a brief discussion on adopting adaptive learning-rate[143] and adaptive weighting scheme[91] with the proposed activation function. PINN-based forward and inverse problems validate the proposed activation on several cases in Section 3.3. Section 3.4 provides some discussion on the results. Finally, Section 3.5 presents the conclusions and future work.

3.1 Physics-informed Neural Networks

Let $\mathbf{x} \in \Omega \subset \mathbb{R}^d$ denote the spatio-temporal location and $u(\mathbf{x}) \in \mathbb{R} \; \forall \; \mathbf{x} \in \Omega$ denote the solution[1] to be determined from an Ordinary Differential Equation (ODE) or Partial Differential

[1]The paper considers $u(\mathbf{x})$ as a scalar field for the sake of simplicity. For the general case, $u(\mathbf{x})$ can be a vector.

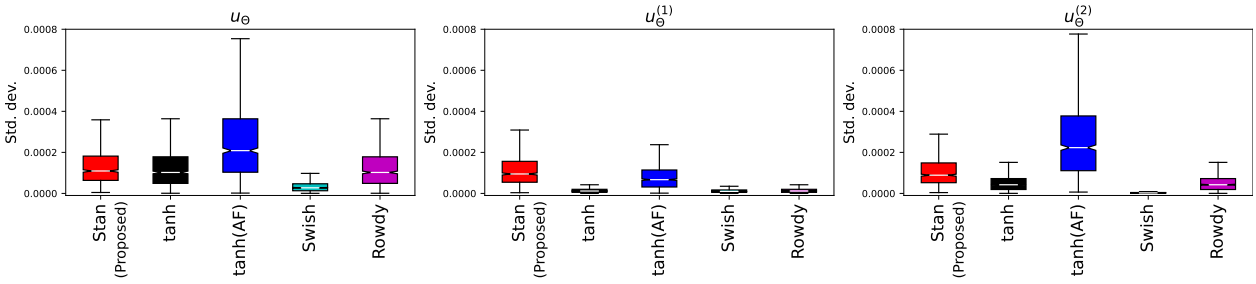


Figure 3.1: The distribution of standard deviation in the computation of $u_{\mathbf{\Theta}}$, $u^{(1)}_{\mathbf{\Theta}}$, and $u^{(2)}_{\mathbf{\Theta}}$ across 100 evenly spaced x values (in [-1,1]) from 1000 different initializations for all the standard activation functions in PINN (2-layer 50-units width NN). Notice the shrinkage in standard deviation for most activation functions in calculating the first derivative.

Equation (PDE) system

$$\begin{aligned} \mathcal{L}[u(\mathbf{x})] &= f(\mathbf{x}),\ \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= g_1(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_1, \\ \nabla u(\mathbf{x}) &= g_2(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_2; \end{aligned} \tag{3.1}$$

where $\mathcal{L}$ represents a linear/nonlinear differential operator, ∇ denotes the gradient operator, f, g_1, and g_2 are known functions, and u is the unknown solution to be learned. $u(\mathbf{x}) = g_1(\mathbf{x})$, $\mathbf{x} \in \partial\Omega_1$ and $\nabla u(\mathbf{x}) = g_2(\mathbf{x})$, $\mathbf{x} \in \partial\Omega_2$ are the Dirichlet and Neumann boundary conditions (DBC and NBC), respectively. To find the solution $u(\mathbf{x})$, Section 3.1.1 introduces the training of the Physics-informed Neural Networks (PINN) [112]. Section 3.1.2 subsequently describes the proposed activation function.

3.1.1 Training

Consider a fully-connected NN of depth $D+1$ corresponding to a network with d-dimensional inputs, D hidden layers, and an output layer. Assume all the hidden layers have P neurons each. $\mathbf{x} \in \Omega$ is the input to the NN, and the NN approximation of the solution function $u(\mathbf{x})$

is the output. For a given input $\mathbf{x}$, the fully connected NN can be represented as follows.

$$\begin{aligned} \mathbf{L}_1 &= \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \\ \mathbf{z}_k &= \sigma(\mathbf{L}_k) \ \forall \ k = \{1, 2, ..., D\} \\ \mathbf{L}_k &= \mathbf{W}_k\mathbf{z}_{k-1} + \mathbf{b}_k \ \forall \ k = \{2, 3, ..., D\} \\ u_{\mathbf{\Theta}}(\mathbf{x}) &= \mathbf{W}_{D+1}\mathbf{z}_D + \mathbf{b}_{D+1} \end{aligned} \tag{3.2}$$

where $\mathbf{W}_k \ \forall \ k \in \{1, 2, ..., D+1\}$ represent the weight matrices and $\mathbf{b}_k \ \forall \ k \in \{1, 2, ..., D+1\}$ represent the bias vectors of corresponding layers in the NN. Note that $\mathbf{W}_k \in \mathbb{R}^{P\times P} \ \forall \ k \in \{2, ..., D\}$, $\mathbf{b}_k \in \mathbb{R}^{P\times 1} \ \forall \ k \in \{1, 2, ..., D\}$, $\mathbf{W}_1 \in \mathbb{R}^{P\times d}$, and for a scalar output $u(\mathbf{x})$, $\mathbf{W}_{D+1} \in \mathbb{R}^{1\times P}$ and $\mathbf{b}_{D+1} \in \mathbb{R}^{1\times 1}$. $\mathbf{z}_k \in \mathbb{R}^{P\times 1} \ \forall \ k \in \{1, 2, ..., D\}$ is the output vector of k^{th} hidden layer of the network and input to the next layer. $\sigma(\cdot)$ is the activation function that acts element-wise on vector $\mathbf{L}_k \in \mathbb{R}^{P\times 1} \ \forall \ k \in \{1, 2, ..., D\}$. $\mathbf{\Theta}$ collectively denotes all the trainable weights and biases.

Learning the solution to Eq. (3.1) implies $u_{\mathbf{\Theta}^*}(\mathbf{x}) \approx u(\mathbf{x})$, where $\mathbf{\Theta}^*$ is the optimal value of $\mathbf{\Theta}$. Choosing three finite sets of points, $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$, such that $\mathcal{F} \subset \Omega$, $\mathcal{U}_1 \subset \partial\Omega_1$, and $\mathcal{U}_2 \subset \partial\Omega_2$ aids in learning the $\mathbf{\Theta}^*$. These sets define the loss function as

$$J(\mathbf{\Theta}) = w_{\mathcal{F}}\text{MSE}_{\mathcal{F}} + w_{\mathcal{U}_1}\text{MSE}_{\mathcal{U}_1} + w_{\mathcal{U}_2}\text{MSE}_{\mathcal{U}_2}, \tag{3.3}$$

where

$$\begin{aligned} \text{MSE}_{\mathcal{F}} &= \frac{1}{|\mathcal{F}|}\sum_{\mathbf{x}\in\mathcal{F}} |\mathcal{L}[u_{\mathbf{\Theta}}(\mathbf{x})] - f(\mathbf{x})|^2, \\ \text{MSE}_{\mathcal{U}_1} &= \frac{1}{|\mathcal{U}_1|}\sum_{\mathbf{x}\in\mathcal{U}_1} |u_{\mathbf{\Theta}}(\mathbf{x}) - g_1(\mathbf{x})|^2, \text{and} \\ \text{MSE}_{\mathcal{U}_2} &= \frac{1}{|\mathcal{U}_2|}\sum_{\mathbf{x}\in\mathcal{U}_2} |\nabla u_{\mathbf{\Theta}}(\mathbf{x}) - g_2(\mathbf{x})|^2, \end{aligned}$$

and $w_{\mathcal{F}}$, $w_{\mathcal{U}_1}$, and $w_{\mathcal{U}_2}$ are the corresponding weights for each segment of the loss function. The points in the set $\mathcal{F}$ are the PDE/ODE residual points or the collocation points since the residual of the PDE/ODE, i.e., $(\mathcal{L}[u_{\boldsymbol{\Theta}}(\mathbf{x})] - f(\mathbf{x}))$ uses these points for calculation. Likewise, $\text{MSE}_{\mathcal{F}}$ is the `PDE/ODE residual loss`. $\text{MSE}_{\mathcal{U}_1}$ is the `Dirichlet Boundary loss` or simply `Data loss`, since these are the only directly available data in the form of inputs and outputs to train the PINN. Also, note that any available data adds to `Data loss` part of the loss function, relaxing the condition that this part should be from the boundary ($\partial\Omega_1$). This relaxation makes the PINN more flexible than the FEM and capable of solving inverse problems. The third part of the loss function $\text{MSE}_{\mathcal{U}_2}$ is the `Neumann Boundary loss`.

The optimal set of parameters $\boldsymbol{\Theta}^*$ minimizes the loss function in Eq. (3.3). To solve the system in Eq. (3.1), all the chosen points in the corresponding sets $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$ should satisfy the corresponding conditions in Eq. (3.1), and thus should minimize the mean-squared errors. Hence, the training to solve the PDE/ODE system involves finding $\boldsymbol{\Theta}^*$ where

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}). \tag{3.4}$$

One of the forms of gradient descent algorithm, Adam[68] or L-BFGS[19] approximates the solution to the problem in Eq. (3.4) iteratively. Figure 1.1 shows the schematic for the PINN training.

3.1.2 Issues with Training

Further elucidation of the training procedure and motivation of the proposed activation function is set forth by a simple second-order ODE system. The problem is

$$\begin{aligned} \frac{d^2u}{dx^2} + \frac{du}{dx} - 6u = 0,\ x \in [0,2], \\ u(0) = 2, \\ \frac{du}{dx}|_{x=0} = -1. \end{aligned} \tag{3.5}$$

$\mathcal{U}_1$ and $\mathcal{U}_2$ are singleton sets in this problem since the corresponding boundary conditions are available only at $x = 0$. Assume N_f number of points are chosen randomly from the domain $[0, 2]$ for the set $\mathcal{F}$. For sake of simplicity, let $w_{\mathcal{F}} = w_{\mathcal{U}_1} = w_{\mathcal{U}_2} = 1$. Thus, the sum of

$$\begin{aligned} \text{MSE}_{\mathcal{F}} =& \frac{1}{N_f}\sum_{i=1}^{N_f}\Big(\frac{d^2u_{\boldsymbol{\Theta}}}{dx^2}|_{x_i} + \frac{du_{\boldsymbol{\Theta}}}{dx}|_{x_i} - 6u_{\boldsymbol{\Theta}}(x_i)\Big)^2 \\ \text{MSE}_{\mathcal{U}_1} =& \Big(u_{\boldsymbol{\Theta}}(0) - 2\Big)^2 \\ \text{MSE}_{\mathcal{U}_2} =& \Big(\frac{du_{\boldsymbol{\Theta}}}{dx}|_{x=0} - (-1)\Big)^2 \end{aligned} \tag{3.6}$$

gives the loss function.

Constructing the loss function in Eq. (3.6) requires the first and the second order derivatives of the output $u_{\boldsymbol{\Theta}}$ with respect to the independent (input) variable x (i.e., $\frac{du_{\boldsymbol{\Theta}}}{dx}$ and $\frac{d^2u_{\boldsymbol{\Theta}}}{dx^2}$, respectively). For instance, $\frac{du_{\boldsymbol{\Theta}}}{dx}|_{x=x_i}$ at some x_i can be calculated by Automatic Differentiation (AD) [10] at any given training step. AD uses the chain rule of differentiation considering the NN parameters $\boldsymbol{\Theta}$ as constant at the given training step.

To make the arguments uncomplicated, this sub-section continue the usage of NN with one-dimensional input and one-dimensional output. Higher-dimensional problems can adopt the

same arguments. Let $u_{\mathbf{\Theta}}^{(1)}$ and $u_{\mathbf{\Theta}}^{(2)}$ denote the first and the second derivatives (i.e., $\frac{du_{\mathbf{\Theta}}}{dx}$ and $\frac{d^2u_{\mathbf{\Theta}}}{dx^2}$), respectively. This work sticks to differential equations with orders less than or equal to 2. For disambiguation, hereon, the word "derivative" typically refers to the partial derivative of output of the NN with respect to the input of the NN (i.e., $u_{\mathbf{\Theta}}^{(1)}$ and $u_{\mathbf{\Theta}}^{(2)}$) at constant NN parameters, and the word "gradient" refers to the partial derivative with respect to the NN parameters with constant x in the typical sense of DL (i.e., for example, $\frac{\partial J(\mathbf{\Theta})}{\partial \mathbf{\Theta}}$ and $\frac{\partial u_{\mathbf{\Theta}}^{(2)}}{\partial \mathbf{\Theta}}$).

Calculating the Derivatives

As in Eq. (3.6), most of the information to train the PINN comes in the form of derivatives. Since the PINN hinge-upon the Universal Approximation Theorem[57, 112], it is easy to imagine that the derivatives of NN should be as expressive as the NN itself. But there are some practical concerns about the expressivity of derivatives to train the NN. To delve deeper into this concern, it is imperative to derive the expression for the calculation of derivatives (i.e., $\frac{du_{\mathbf{\Theta}}}{dx}$ and $\frac{d^2u_{\mathbf{\Theta}}}{dx^2}$).

Lemma 3.1. *For the NN in Eq. (3.2), Eq. (3.7) provides the expression for calculating the first derivative of a single-input and single-output NN in closed form at any x (derivation is in Appendix B.1).*

$$\begin{aligned} \mathbf{H}_1 &= \mathbf{W}_1 \\ \mathbf{G}_i &= \sigma^{'}(\mathbf{L}_i) \circledcirc \mathbf{H}_i \ \forall \ i \in \{1, 2, 3, ..., D\} \\ \mathbf{H}_i &= \mathbf{W}_i \mathbf{G}_{i-1} \ \forall \ i \in \{2, 3, ..., D+1\} \\ u_{\mathbf{\Theta}}^{(1)}(x) &= \mathbf{H}_{D+1} \end{aligned} \tag{3.7}$$

where $\sigma^{'}$ is an element-wise operator which gives the derivative of the activation function

at $\mathbf{L}_k^p \ \forall \ k \in \{1, 2, 3, ..., D\}, \ \forall \ p \in \{1, 2, 3, ..., P\}$ *and '*$\circledcirc$*' is the element-wise multiplication operator.* $\mathbf{H}_i \in \mathbb{R}^{P\times 1}(\ \forall \ i \in \{1, 2, 3, ..., D\})$ *and* $\mathbf{G}_i \in \mathbb{R}^{P\times 1}(\ \forall \ i \in \{1, 2, 3, ..., D\})$ *are intermediate variables to show the iterative calculation.* $\mathbf{H}_{D+1} \in \mathbb{R}$ *for one-dimensional* $u_{\mathbf{\Theta}}$ *and one-dimensional* x*.*

In Eq. (3.7), note that $\sigma^{'}$ is the only source of non-linearity in calculating $u_{\mathbf{\Theta}}^{(1)}$. These $\sigma^{'}$ terms are the same terms that are encountered in the backpropagation of a purely data-driven loss function. For instance, consider the $\text{MSE}_{\mathcal{U}_1}$ in Eq. (3.6),

$$\frac{\partial \text{MSE}_{\mathcal{U}_1}}{\partial \theta} = 2 \times \Big(u_{\mathbf{\Theta}}(0) - 2\Big) \times \frac{\partial u_{\mathbf{\Theta}}(0)}{\partial \theta} \tag{3.8}$$

for some NN parameter θ. If θ is one of NN weights, then the gradient of θ in the intermediate layers evaluates to Eq. (3.10) (backpropagation).

$$\begin{aligned} \mathbf{\Lambda}_1 &= \mathbf{W}_{D+1} \\ \mathbf{\Upsilon}_i &= \sigma^{'}(\mathbf{L}_{D-i}) \circledcirc \mathbf{\Lambda}_i \ \forall \ i \in \{1, 2, 3, ..., D\} \\ \mathbf{\Lambda}_i &= \mathbf{W}_{D+1-i}\mathbf{\Upsilon}_{i-1} \ \forall \ i \in \{2, 3, ..., D\} \end{aligned} \tag{3.9}$$

$$\frac{\partial u_{\mathbf{\Theta}}}{\partial \mathbf{W}_{D-i}^{qp}} = \mathbf{z}_{D-i-1}^{q}\mathbf{\Upsilon}_{i+1}^{p} \ \forall \ i \in \{1, 2, ..., D-1\} \tag{3.10}$$

$\mathbf{\Lambda}_i \in \mathbb{R}^{P\times 1}(\ \forall \ i \in \{1, 2, 3, ..., D\})$ and $\mathbf{\Upsilon}_i \in \mathbb{R}^{P\times 1}(\ \forall \ i \in \{1, 2, 3, ..., D\})$ are intermediate variables to show the iterative calculation.

It is easy to see the similarity between Eq. (3.9) and Eq. (3.7) (note that $\left|\mathbf{z}_{D-i-1}^{q}\right|$ is less than 1 for the tanh function). Evidently, the success of the tanh as an activation function rests on its property that it can evaluate to normalized values close to zero[42, 75], and hence these

$\sigma^{'}$ terms are close to one for effective gradient computation (Eq. (3.9) and Eq. (3.10)). As expected, the derivative of the tanh closer to one provides robust gradients to train a purely data-driven NN[75]. The following analysis shows that the same property of the tanh can be problematic for PINN. Let the tanh function evaluate to a normal distribution with zero mean and s^2 variance. It is straightforward to show that (derivation is in Appendix B.2), if σ is tanh,

$$\begin{aligned} \mathbb{E}[\sigma^{'}] &= 1 - s^2 \\ \mathbb{V}[\sigma^{'}] &= 2s^4 \end{aligned} \tag{3.11}$$

Perhaps, s can be low, say 0.3, since the tanh is restricted between -1 and 1. In this case, the expected value of $\sigma^{'}$ is 0.91, and the variance is 0.0162. As a consequence of $\sigma^{'}$ having mean close to one (also, bounded by one) and low variance, Eq. (3.7) can evaluate to a constant value irrespective of x. The tanh function moving away from this *linear-regime* is necessary for the NN to express more complex functions and hence solve the given differential equation system (successful training in Figure 3.2). At the same time, moving away from the *linear-regime* of the tanh, makes the $\sigma^{'}$ values closer to zero affecting the gradient computation in Eq. (3.9). This dichotomy on the requirements for $\sigma^{'}$ can lead to failed training of PINN.

The second derivative of the NN does not have this issue due to a higher variance of $\sigma^{''}$ at the *linear-regime* as further explained by Lemma 3.2.

Lemma 3.2. *As a follow-up of Lemma 3.1, the equation below provides the second derivative*

of the NN at any x.

$$\begin{aligned}
\mathbf{F}_1 &= \mathbf{0} \\
\mathbf{C}_i &= \sigma^{''}(\mathbf{L_i}) \circledcirc \mathbf{H}_i \circledcirc \mathbf{H}_i \ \forall\ i \in \{1, 2, 3, ..., D\} \\
\mathbf{E}_i &= \mathbf{C}_i + \sigma^{'}(\mathbf{L_i}) \circledcirc \mathbf{F}_i \ \forall\ i \in \{1, 2, 3, ..., D\} \\
\mathbf{F}_i &= \mathbf{W}_i \mathbf{E}_{i-1} \ \forall\ i \in \{2, 3, ..., D+1\} \\
u^{(2)}_{\mathbf{\Theta}}(x) &= \mathbf{F}_{D+1}
\end{aligned} \tag{3.12}$$

where $\sigma^{''}$ is an element-wise operator which gives the derivative of the activation function at $L^p_k\ \forall\ k \in \{1, 2, 3, ..., D\},\ \forall\ p \in \{1, 2, 3, ..., P\}$ and '$\circledcirc$' is the element-wise multiplication operator. $\mathbf{C}_i \in \mathbb{R}^{P\times 1}(\ \forall\ i \in \{1, 2, 3, ..., D\})$, $\mathbf{E}_i \in \mathbb{R}^{P\times 1}(\ \forall\ i \in \{1, 2, 3, ..., D\})$ and $\mathbf{F}_i \in \mathbb{R}^{P\times 1}(\ \forall\ i \in \{1, 2, 3, ..., D\})$ are intermediate variables to show the iterative calculation. $\mathbf{F}_{D+1} \in \mathbb{R}$ for one-dimensional $u_{\mathbf{\Theta}}$ and one-dimensional x.

Consider $\sigma^{''}$ as the second derivative of the tanh function (derivation is in Appendix B.3),

$$\begin{aligned}
\mathbb{E}[\sigma^{''}] &= 0 \\
\mathbb{V}[\sigma^{''}] &= 4s^2(15s^4 - 6s^2 + 1).
\end{aligned} \tag{3.13}$$

For the same value of $s = 0.3$, the variance of $\sigma^{''}$ is approximately 0.21, which is significantly higher than the variance of $\sigma^{'}$ (and σ). As a consequence, the second derivative is better expressed than the first derivative for the tanh function.

The $\sigma^{'}$ values around 1 for the tanh causes a shrinkage in expressiveness of the first derivative $u^{(1)}_{\mathbf{\Theta}}$ of NN as compared to the NN's ($u_{\mathbf{\Theta}}$) expressiveness for a typical Glorot normal initialization [42] (as in Figure 3.1). The shrinkage in expressiveness at initialization is observed across the literature activation functions. The improved standard deviation for the

calculation of $u_{\mathbf{\Theta}}^{(2)}$ for the tanh activation function can be attributed to the improved variance in Eq. (3.13). Further, Figure 3.2 compares a successful and unsuccessful case in PINN training to substantiate the argument on the tanh function.

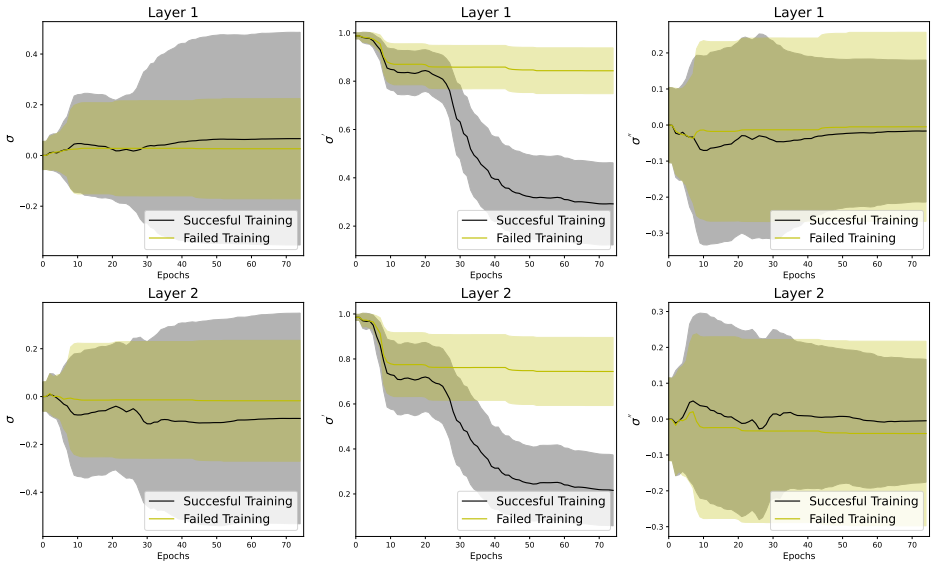


Figure 3.2: Comparison of a successful training and an unsuccessful training on second-order ODE with the tanh activation function (2-layer 50-units width NN). The mean of the activation function, its first, and its second derivative values are plotted during the course of training, along with one standard deviation error-bar. Note the distributions of σ and $\sigma^{''}$ are not as vastly different as the distribution of $\sigma^{'}$ between the two cases.

Adaptive tanh [60, 61] partially overcomes this issue by multiplying an extra trainable coefficient to the input of the tanh function (which can increase the variance of $\sigma^{'}$). Rowdy activation [63] overcomes the same by adding several sinusoidal terms along with the extra trainable coefficient to the input of the tanh function. The Swish function[114] empirically seem to be affected by shrinkage as well. Appendix B.4 provides a visualization of the distribution of the activation functions and its first and second derivatives during training.

Flow of Gradients

Learning the solution of any NN happens through the gradients of the loss function. The most intriguing part of the loss function is $\text{MSE}_{\mathcal{F}}$. Analyzing the gradient of $\text{MSE}_{\mathcal{F}}$ is the foremost task in studying the training of PINN.

$$\begin{aligned}\frac{\partial \text{MSE}_{\mathcal{F}}(\mathbf{\Theta})}{\partial \theta} =& \frac{2}{N_f}\sum_{i=1}^{N_f}\Big(u^{(2)}_{\mathbf{\Theta}}(x_i) + u^{(1)}_{\mathbf{\Theta}}(x_i) - 6u_{\mathbf{\Theta}}(x_i)\Big)\times \\ & \frac{\partial\Big(u^{(2)}_{\mathbf{\Theta}}(x_i) + u^{(1)}_{\mathbf{\Theta}}(x_i) - 6u_{\mathbf{\Theta}}(x_i)\Big)}{\partial\theta} \\ \frac{\partial \text{MSE}_{\mathcal{F}}(\mathbf{\Theta})}{\partial \theta} =& \frac{2}{N_f}\sum_{i=1}^{N_f}\Big(u^{(2)}_{\mathbf{\Theta}}(x_i) + u^{(1)}_{\mathbf{\Theta}}(x_i) - 6u_{\mathbf{\Theta}}(x_i)\Big)\times \\ & \Big(\frac{\partial u^{(2)}_{\mathbf{\Theta}}(x_i)}{\partial\theta} + \frac{\partial u^{(1)}_{\mathbf{\Theta}}(x_i)}{\partial\theta} - 6\frac{\partial u_{\mathbf{\Theta}}(x_i)}{\partial\theta}\Big)\end{aligned} \tag{3.14}$$

Inspecting the particular form of the loss function in Eq. (3.14) is hard and perhaps useless for extending to other differential equation systems. Instead, the terms $\frac{\partial u^{(2)}_{\mathbf{\Theta}}(x_i)}{\partial\theta}$, $\frac{\partial u^{(1)}_{\mathbf{\Theta}}(x_i)}{\partial\theta}$, and $\frac{\partial u_{\mathbf{\Theta}}(x_i)}{\partial\theta}$ that determine the gradient of the loss function irrespective of the particular form of differential equation can be more useful. This section provides some insights into these gradient terms to validate the appropriateness of the activation functions for PINN.

To keep the arguments simple, for a single-layer NN, Eq. (3.7) reduces to

$$u^{(1)}_{\mathbf{\Theta}}(x) = \sum_{i=1}^{P}\mathbf{W}^i_2.\mathbf{W}^i_1.\sigma'(\mathbf{W}^i_1 x + \mathbf{b}^i_1). \tag{3.15}$$

For the single hidden-layer derivative expression in Eq. (3.15),

$$\begin{aligned}\frac{\partial u^{(1)}_{\mathbf{\Theta}}(x)}{\partial \mathbf{W}^p_2} &= \mathbf{W}^p_1.\sigma^{'}(\mathbf{W}^p_1 x + \mathbf{b}^i_1) \\ \frac{\partial u^{(1)}_{\mathbf{\Theta}}(x)}{\partial \mathbf{W}^p_1} &= \mathbf{W}^p_2(\mathbf{W}^p_1)^2\sigma^{''}(\mathbf{W}^p_1 x + \mathbf{b}^i_1) + \mathbf{W}^p_2\sigma^{'}(\mathbf{W}^p_1 x + \mathbf{b}^i_1)\end{aligned} \tag{3.16}$$

From Eq. (3.16), the gradient of weights in the first layer depend on the second derivative of the activation function at that layer. As the tanh supposedly operates in the *linear-regime*[42], from the second derivative's expectation value of zero in Eq. (3.13), the loss gradients can be poor. Therefore, the tanh function can cause poor training in learning the first derivative of the target function. Figure 3.3 shows a way to compare the activation functions in regards to Eq. (3.16).

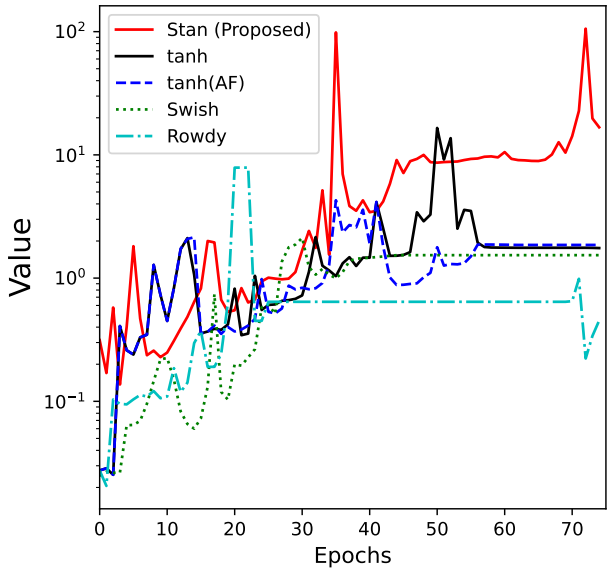


Figure 3.3: Average of $\frac{\left|\mathbf{W}^p_2(\mathbf{W}^p_1)^2\sigma^{''}(\mathbf{W}^p_1 x+\mathbf{b}^p_1)\right|}{\left|\frac{\partial u^{(1)}_{\mathbf{\Theta}}(x)}{\partial \mathbf{W}^p_1}\right|}$ at some x over 50 neurons for training a single layer PINN for the motivational second-order ODE problem. Higher values imply better contribution of the second derivative term in Eq. (3.16).

Example 3.3. This example shows that the tanh activation function exaggerates the spectral-

bias of PINNs (refer to[100, 144, 145]). Moseley et al. [100] used a domain-decomposition approach and Wang et al. [144] suggested a novel network architecture to overcome the spectral-bias of PINNs. Indeed, spectral-bias is a well-known issue of any general NN[110, 147]. The motivating example in Moseley et al.'s[100] work inspired the considered first-order ODE.

Consider modeling the function $u = \sin(10x),\ x \in [-1, 1]$ using traditional NN with the tanh activation function using sufficient data. Observe that the NN can model the function pretty well (Figure 3.4), resulting in an MSE of $2.6e-4$, ruling out the possibility of spectral-bias in NN itself.

Now, use the derivative information $\frac{du}{dx} = 10\cos(10x)$, and a single data point to do a PINN training on the same architecture and activation function. The points of set $\mathcal{F}$ are the same as the data points. Note the significant drop in performance (MSE $= 0.46$) as clear from the figure.

Further, to confirm the role of the tanh activation function in the failed training, simply add a linear term (with coefficient 1) to the activation function as suggested in the key literature on the tanh activation function[75]. Though the second derivative does not change with the additional linear term to the activation function, the expressivity of the first derivative (as discussed in Section 3.1.2) does not change either. This addition improves the performance drastically (MSE $= 2.4e-3$). Conceivably, the improvement is due to the extra weight (of value 1) added to the first derivative term in Eq. (3.16).

Again for a single-layer NN, Eq. (3.17) gives the second derivative of the NN and Eq. (3.18) gives the second derivative's gradients.

$$u^{(2)}_{\mathbf{\Theta}}(x) = \sum_{i=1}^{P} \mathbf{W}^i_2.(\mathbf{W}^i_1)^2.\sigma^{''}(\mathbf{W}^i_1 x + \mathbf{b}^i_1) \tag{3.17}$$

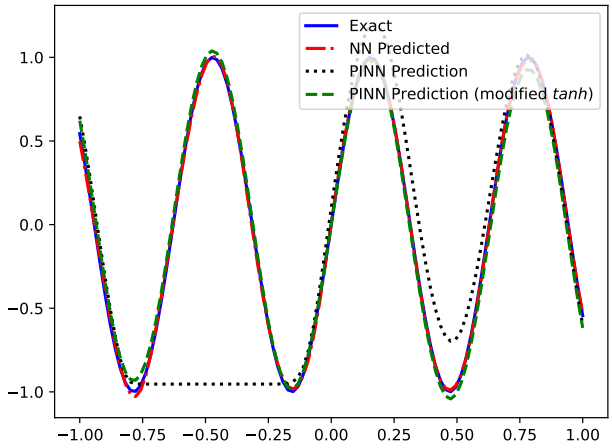


Figure 3.4: Comparing the predictions of a PINN training with and without modifiying the tanh activation function and prediction of a data-driven regular NN of the same sinusoidal function.

$$\begin{aligned}\frac{\partial u^{(2)}_{\mathbf{\Theta}}(x)}{\partial \mathbf{W}^p_2} &= (\mathbf{W}^p_1)^2\sigma''(\mathbf{W}^p_1 x + \mathbf{b}^i_1)\\ \frac{\partial u^{(2)}_{\mathbf{\Theta}}(x)}{\partial \mathbf{W}^p_1} &= \mathbf{W}^p_2(\mathbf{W}^p_1)^3\sigma'''(\mathbf{W}^p_1 x + \mathbf{b}^i_1)\\ &\quad +2\mathbf{W}^p_2\mathbf{W}^p_1\sigma''(\mathbf{W}^p_1 x + \mathbf{b}^i_1)\end{aligned} \tag{3.18}$$

σ''' is the third derivative of the activation function. For the normal assumption of the tanh function (with $s = 0.3$ in Section 3.1.2), the expectation of the third derivative is around -1.43. Therefore, there is no issue with third derivative in Eq. (3.18), whereas the gradient issue persists because of the second derivative. These arguments on gradients can be extended to multi-layer NN as well (see Appendix B.4)

Though the adaptive activation function[60, 61] may follow a similar trend in the activation function's derivatives, Swish[114] and Rowdy[63] activation functions may not. For the latter two concerned activation functions, this work resorts to empirical results (in Section 3.2 and Section 3.3).

Output Scaling

Another identified reason for failure of PINN, is the unknown output range. Though this issue is particular to PINNs, the reasoning does not involve solving differential equations using the unconventional loss function. In a purely data-driven NN, irrespective of the depth of NN, the gradient of weights of the last layer depends on just the outputs of the penultimate layer.

$$\frac{\partial u_{\mathbf{\Theta}}}{\partial \mathbf{W}^p_{D+1}} = \mathbf{z}^p_D \tag{3.19}$$

In Eq. (3.19), $\mathbf{z}^p_D$ is the output of the penultimate layer. For the tanh and its adaptive modification[60, 61], $\mathbf{z}^p_D$ is restricted between -1 and 1. The lower magnitude (compared to the magnitude of the solution of the differential equation) of $\mathbf{z}^p_D$ implies ineffective gradients for the final layer weights.

For example, the motivational problem in Eq. (3.5) has the analytical solution $u(x) = \exp(2x) + \exp(-3x)$. The function $u(x)$ takes a minimum value of 2 at $x = 0$ and a maximum value of 54.6 at $x = 2$. If this solution function were to be modeled with an NN provided sufficient data, the usual practice is to scale down the data such that it is normalized or standardized[75]. Though the last layer is typically linear for regression problems, scaling-down makes the function converge faster[75]. Scaling down requires statistics of the provided data. For PINN, as apparent from the boundary conditions in Eq. (3.5), it is impossible to know the range of values the solution $u(x)$ might take. As the converse of scaling down the data accelerating convergence, the unscaled output might slow down the convergence.

In PINN, the effect of output scaling might not be limited to a mere slowdown in the convergence. It is more likely to affect the derivative and gradient computations subsequently

making the loss get stuck. From Lemma 3.1 and Lemma 3.2, the role of the last layer weights ($\mathbf{W}_{D+1}$) is apparent in the derivative calculation. All the gradients of weights from the first layer to the penultimate layer depend on the final layer's weights (Eqs. (3.9), (3.16), and Appendix B.4). All the numerical studies (especially the regression problem in Section 3.2.1) and case studies testify the argument.

Swish and Rowdy activation functions [63, 114] are possibly immune to this issue due to unbounded output. Also, the scaling problem can be avoided altogether for certain forms of differential equations using the non-dimensionalization trick [13].

3.1.3 Proposed Self-scalable Tanh Activation Function

Though most of the theoretical arguments in the previous sections centered around the tanh function, Figure 3.1 and Figure 3.3 show that the arguments can be extended to other activation functions of PINN as well. After all, the tanh function remains the most sought-after activation function for PINN. Adaptive tanh function[60, 61] is a trainable version of a modification proposed by LeCun et al. [75]. Another modification provided by LeCun et al. [75], adding a linear term, is used in Example 3.3. The modification with a linear term still does not address the derivative expressivity issue discussed in Section 3.1.2.

To summarize the lessons from previous sections,

1. The expressivity of the first derivative ($u^{(1)}_{\mathbf{\Theta}}$) should not be detrimental to the gradient calculations and vice versa.

2. The second derivative of the activation function should not concentrate around zero.

3. The magnitude of the output of the activation function should not be restricted too much.

Naturally, LeCun et al.'s addition of a linear term[75] provides a base to build on. Accordingly, the proposed Self-scalable tanh (Stan) activation function for the i^{th} neuron in k^{th} layer ($k = 1, 2, \ldots, D-1;\ i = 1, 2, \ldots, N_k$) is

$$\sigma_k^i(x) = (1 + \beta_k^i x) \cdot \tanh(x), \tag{3.20}$$

where $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and β_k^i is the neuron-wise parameter optimized during training along with NN parameters $\boldsymbol{\Theta}$. β_k^i, typically initialized as 1, predominantly controls the expressivity, along with the tanh non-linearity to maintain the gradients when solving higher-order derivatives. Figure 3.5 shows the visualization of Stan activation and its gradients for various β_k^i values. From Figure 3.5(a), note that the *linear-regime* of the tanh is not affected by the proposed modification. Also note from Figure 3.5(b), the maximum of activation function's derivative is not at the *linear-regime*, allowing better expressivity of the first derivative $u_{\boldsymbol{\Theta}}^{(1)}$. Also note that, unlike tanh, the extrema of derivative (i.e., second derivative equals zero) are not at the *linear-regime*.

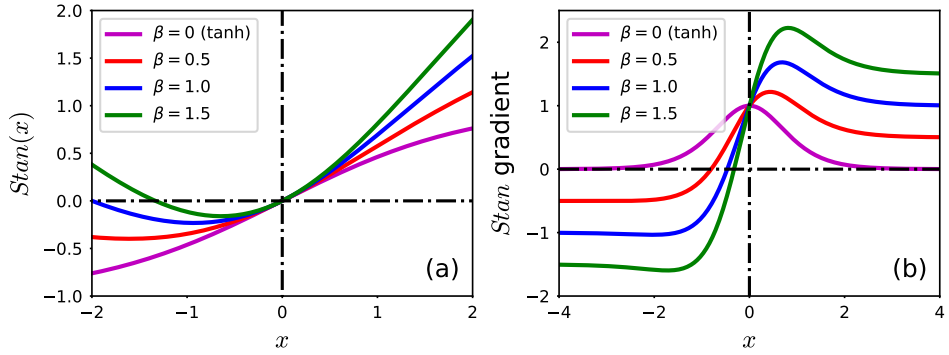


Figure 3.5: Visualization of the proposed Stan activation function in comparison with the tanh. At $x = 0$, the function value is 0 and the gradient is 1. The point of *zero gradient* and *maximum gradient* are dependent on β.

The tanh non-linearity is to make use of its continuity and differentiability, providing a smooth gradient over the entire support[75]. The second term, namely, $\beta_k^i x \cdot \tanh(x)$ is the

self-scaling term that can help to better map different orders of magnitude of the output from the input. As observed from Figure 3.5, the proposed Stan activation function is unbounded above and does not saturate[2] as established by the following proposition as well.

Proposition 3.4. *If $\beta_k^i \neq 0$, the proposed activation function in (3.20) is not a saturation function.*

Proof. The first-order derivative of $\sigma_k^i(x)$ is

$$(\sigma_k^i)'(x) = (1 + \beta_k^i x)\Big(1 - \tanh^2(x)\Big) + \beta_k^i \tanh(x).$$

Therefore, we have

$$\lim_{x \to +\infty} (\sigma_k^i)'(x) = \beta_k^i,$$
$$\lim_{x \to -\infty} (\sigma_k^i)'(x) = -\beta_k^i.$$

□

The proposed Stan contributes additional $\sum_{k=1}^{D-1} N_k$ parameters to optimize. Thus, we have $\tilde{\mathbf{\Theta}} = \{\mathbf{W}^k, \mathbf{b}^k\}_{k=1}^{D} \bigcup \{\beta^k\}_{k=1}^{D-1}$ as trainable parameters, where $\beta^k = (\beta_k^1, \beta_k^2, \ldots, \beta_k^{N_k})^\top$. The expression

$$\tilde{\mathbf{\Theta}}_{m+1} = \tilde{\mathbf{\Theta}}_m - \eta_m \nabla J(\tilde{\mathbf{\Theta}}_m), \tag{3.21}$$

updates the parameters $\tilde{\mathbf{\Theta}}$, where $\eta_m \in (0, 1]$ is the learning rate at m^{th} training iteration.

Convergence

Analyzing the theoretical convergence with the proposed activation function is challenging due to the complicated loss function. Instead, looking at empirical results gives some idea

[2]Saturates only when $\beta_k^i = 0$, which is just the tanh function.

of the loss landscape created by the proposed activation function.

The Standard Condition Number (SCN)[11] provides insight into the loss landscape (low SCN is better). SCN is the ratio of the largest to smallest eigenvalues of Hessian Matrix for the loss function with respect to all the trainable parameters. Consider a first-order ODE system (provided in Eq. (3.23) with $C = 100$). Figure 3.6 shows the SCN values for the first few epochs for various activation functions[3]. Note that the SCN is low for the Stan function over the epochs, similar to the tanh. The results in the next two sections show the benefits of using the Stan activation function over the tanh.

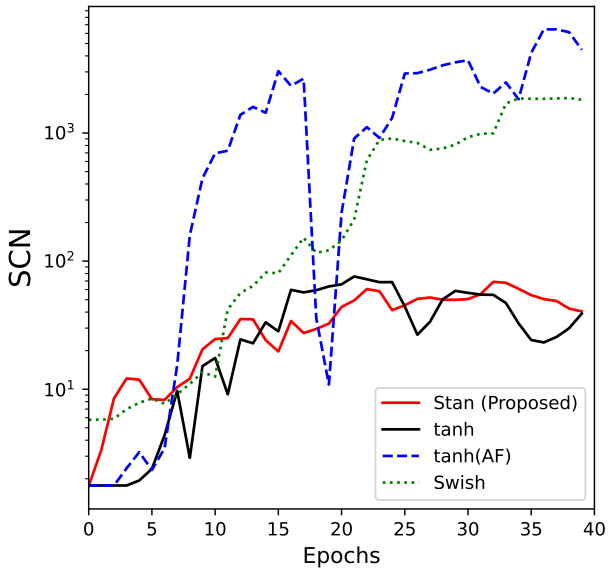


Figure 3.6: The SCN values corresponding to activation functions used in this work for solving a first-order ODE.

3.2 Numerical Studies

A regression problem and two one-dimensional ODE problems described in Section 3.2.1 and Section 3.2.2 demonstrate the effectiveness of the proposed Stan function. Each problem is

[3]Rowdy activation function[63] is skipped due to large number of trainable parameters.

solved at three levels (dubbed as *high*(H), *medium*(M), and *low*(L)) to give a better idea of the significance of the proposed activation function. The three levels, defined individually for each problem, typically point to a high, an intermediate, and a low range of output values.

In all the analyses, the benchmarks for comparison are the state-of-the-art activation functions for PINN, namely Rowdy[63] and Swish[55], along with the tanh and the Neuron-wise Locally Adaptive Activation Function (tanh(AF))[60]. In addition, Section 3.2.3 provides some results and discussions on state-of-the-art methods to accelerate convergence in PINN using adaptive hyperparameters. All the presented results are repeated ten times with different initializations of NN weights and biases and the appropriate sets $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$ to obtain the average performance. The learning rate and other hyper-parameters corresponding to the activation functions are tuned using a grid search of feasible values (see Appendix B.5 for more details). All the codes[4] employ Python 3 with Pytorch 1.9.0 package run on an NVIDIA 2080 Ti GPU.

3.2.1 Neural Network Approximation of Function

For this numerical study, a non-linear discontinuous function highlights particularly the output scaling issue discussed in Section 3.1.2, decoupling the issue from involvement of derivatives. The function definition is

$$u(x) = \begin{cases} C \times (0.2 \sin(6x) \exp(-x)), & -4 \leq x \leq 0 \\ C \times (11 + 0.1x \exp(x)), & 0 < x \leq 3.75, \end{cases} \tag{3.22}$$

[4]The codes are easy to implement as a simple modification to the activation function. Nonetheless, the codes will be made available in a public platform once the paper is published.

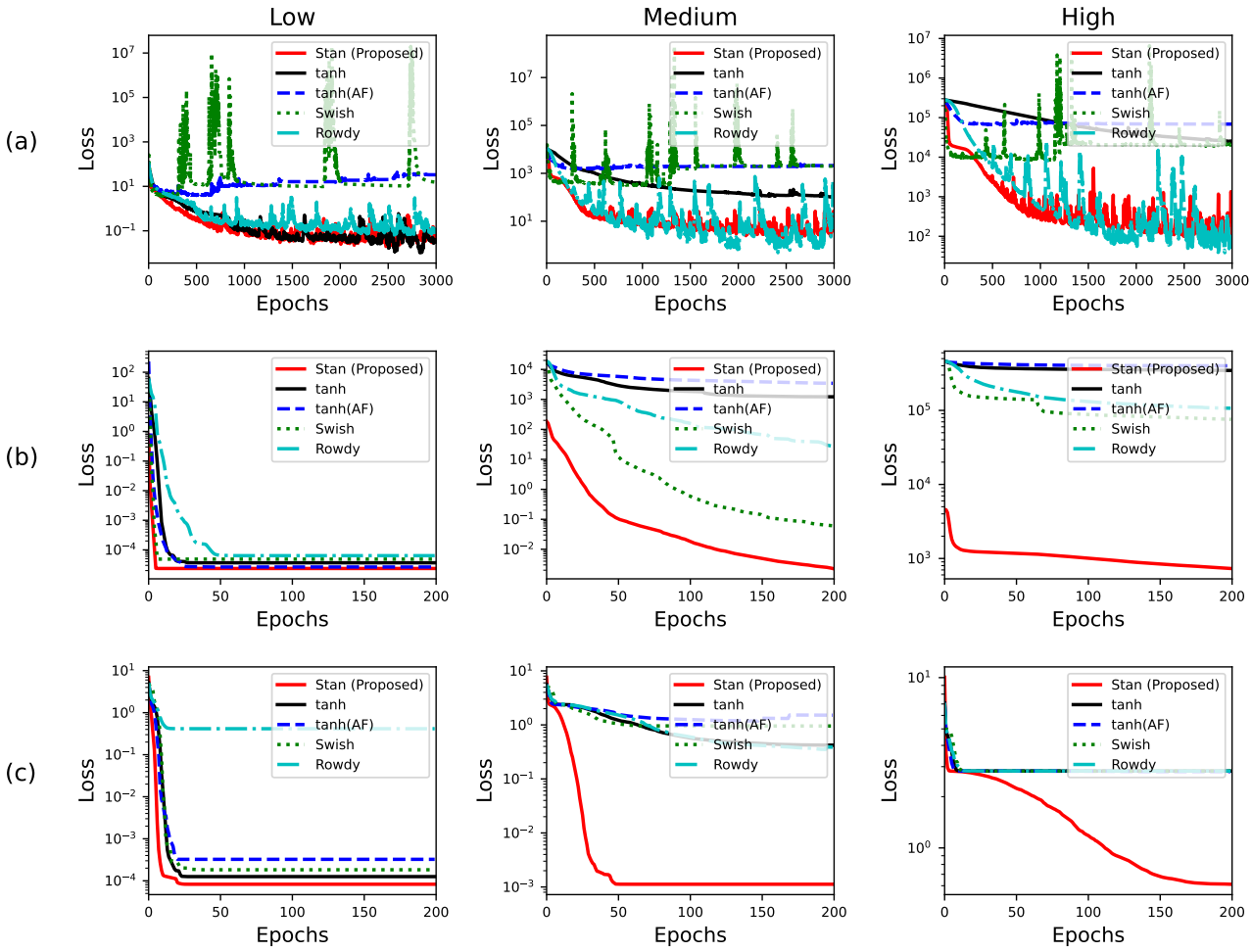


Figure 3.7: Training loss convergence plots averaged over 10 different initializations of weights and biases of the NNs/PINNs with various activation functions for (a) regression problem (b) first-order differential equation, and (c) second-order differential equation at the corresponding three levels *low*, *medium*, and *high*. The convergence of β parameters are provided in Appendix B.10.

which is discontinuous at $x = 0$. Here, C is a constant and taken as 1, 10, and 50 for the three levels *low*, *medium*, and *high*, respectively. Training of the NN models used a set of 300 randomly sampled points with their corresponding $u(x)$. Adam optimizer is employed to train the NNs with four hidden layers of 50 neurons each. The loss function is the mean

squared error of prediction on the training data as given below.

$$\text{MSE}_u = \frac{1}{300}\sum_{i=1}^{300}\left|u_{\tilde{\Theta}}(x_u^i) - u(x_u^i)\right|^2$$

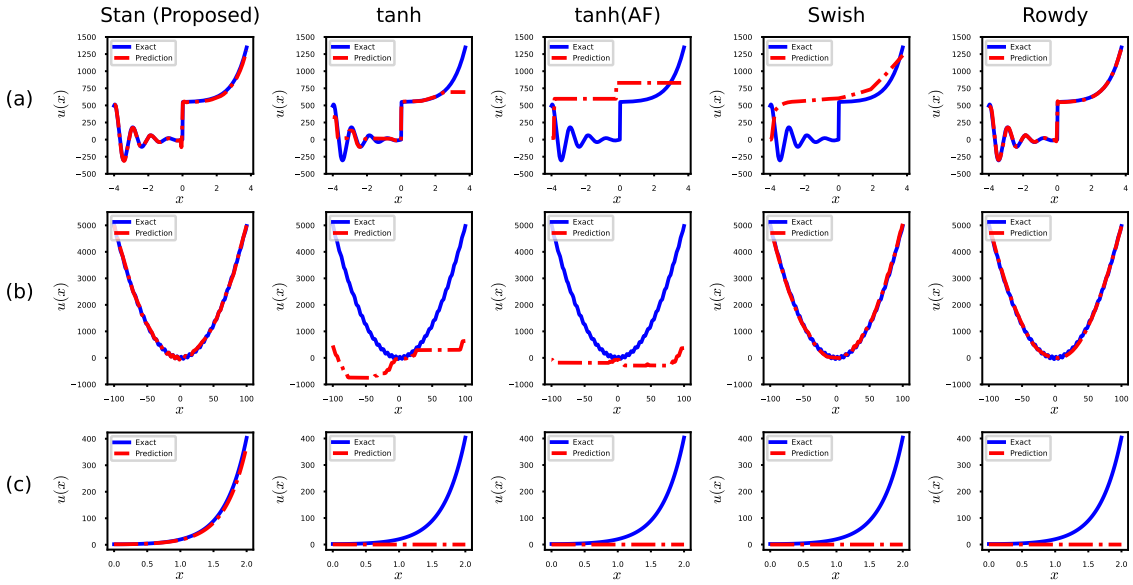


Figure 3.8: Representative predictions of the trained NNs/PINNs with various activation functions in comparison with the exact function for (a) regression problem (b) first-order differential equation, and (c) second-order differential equation at the *high* levels. Appendix B.6 provides predictions at the other levels.

Table 3.1: Test performance from the numerical studies on the regression problem, the first-order ODE, and the second-order ODE in terms of RE. Appendix B.6 provides the same test performance in terms of MSE.

	Regression			First-order ODE			Second-order ODE		
	L	M	H	L	M	H	L	M	H
Stan (Proposed)	**0.04**	**0.05**	**0.05**	**0.00**	**0.00**	**0.02**	**0.00**	**0.00**	**0.25**
tanh	0.05	0.11	0.31	0.00	0.14	0.96	0.00	0.25	1.00
tanh (AF) [60, 61]	0.51	0.41	0.47	0.00	0.60	1.03	0.00	0.95	1.00
Swish [114]	0.33	0.33	0.33	0.00	0.00	0.02	0.00	0.40	1.00
Rowdy [63]	0.06	0.06	0.05	0.00	0.00	0.10	0.10	0.24	1.00

The purposely non-standardized output emphasizes the importance of standardizing the out-

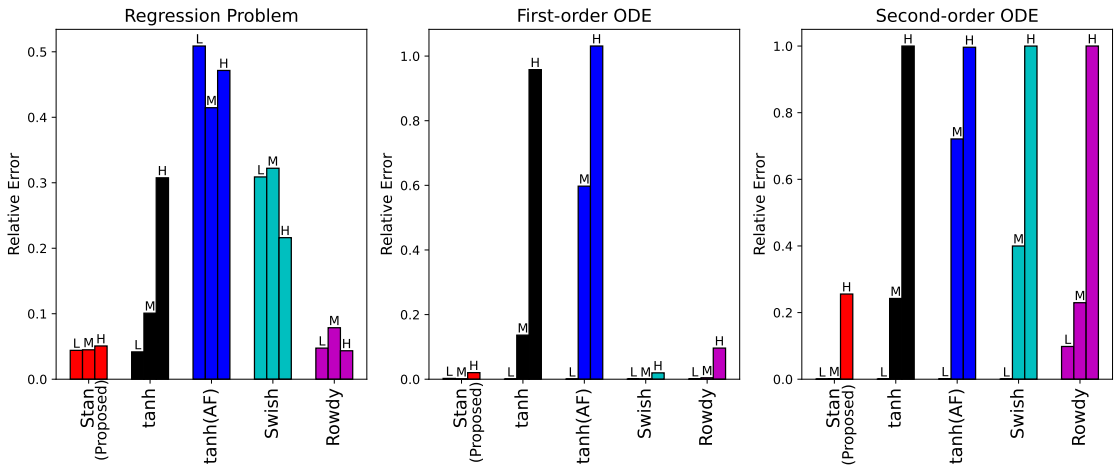


Figure 3.9: Comparison of activation functions on the numerical studies on the regression problem, the first-order ODE problem, and the second-order ODE problem in terms of relative error in prediction.'L' denotes *low*, 'M' denotes *medium*, and 'H' denotes *high* as described in the main text for related problems. The proposed activation function is robust to the magnitude of the output. Table 3.1 provides the values.

put for the traditional activation functions and signifies the suitability of the Stan activation function. Figure 3.7 shows the convergence of the average training loss for all the considered activation functions at all levels. As expected, the tanh is good at the *low* level problem but not so good at higher levels. The Rowdy activation function and the proposed Stan activation function converge to better values than other activation functions irrespective of the level.

Further, the trained models used a set of 1000 evenly spaced points from the support ($-4 \leq x \leq 3.75$) to predict the functions in Eq. (3.22). Figure 3.8(a) provides a representative prediction plot (at the *high* level) for all the models, corresponding to each activation function, along with the exact functional curve. The predicted functional values from the NNs with the Stan and Rowdy activation function closely match the exact function. Table 3.1 summarizes the relative errors (RE) at all the three levels. As seen in Table 3.1 and Figure 3.9, the Rowdy activation function is almost as successful as the proposed Stan function, whereas the other activation functions show poor performance on this problem.

3.2.2 One-dimensional ODEs

In addition to the significance of output scaling, the PINNs suffer from issues related to the calculation of derivatives and gradients, discussed in Section 3.1. Solving first-order and second-order ODE problems with the PINNs underlines the importance of these issues. For PINNs, the L-BFGS optimizer is empirically better than the Adam optimizer.

First-order ODE

The considered first-order ODE is

$$\begin{aligned} &\frac{du}{dx} = C\frac{\cos(x)}{2} + x, \;\; x \in [-C, C], \\ &u(0) = 0. \end{aligned} \tag{3.23}$$

Specifically, considering the problem in Eq. (3.23) with three levels of C values at 1, 20, and 100, the loss function follows the formulation presented in Eq. (3.3) as restated for this problem as

$$\begin{aligned} J(\tilde{\mathbf{\Theta}}) = &\frac{w_F}{10000}\sum_{i=1}^{10000}\left|\frac{du_{\tilde{\mathbf{\Theta}}}}{dx}|_{x_f^i} - C\frac{\cos(x_f^i)}{2} - x_f^i\right|^2 \\ &+ w_u\left|u_{\tilde{\mathbf{\Theta}}}(0) - u(0)\right|^2. \end{aligned} \tag{3.24}$$

The weights w_F and w_u are 100 and 1, respectively, and at every epoch, the training procedure chooses a set of 10,000 residual points. An NN of three hidden layers with 50 neurons in each layer is utilized for training. Unlike other activation functions, the average training loss for NNs with the Stan activation function converges to a much lower value early in training, as shown in Figure 3.7. The low initial loss for Stan function can be attributed to the expressiveness at initialization described in Section 3.1.

The problem in Eq. (3.23) has the known solution $u(x) = C\frac{\sin(x)}{2} + \frac{x^2}{2}$ and testing and comparing the NNs utilized 1000 evenly spaced points in the support. Figure 3.8(b) displays a representative prediction from the PINNs with all the activation functions individually (at the *high* level). As a direct consequence of superior training, the prediction from PINNs with Stan activation functions approximate the exact function extremely well. The data in Table 3.1 shows the wide margin of difference (in terms of RE) between the PINNs with Stan and Swish activation functions and those with other activation functions. Figure 3.9 conveys the same.

Second-order ODE

Increasing the order of the problem further makes it harder for the benchmark activation functions. Solving the representative problem described in Section 3.1.2 provides a clearer picture of the complexity of the PINNs. The problem in Eq. (3.5) is taken to be the *medium* level problem that is defined more generally as

$$\begin{aligned} \frac{d^2u}{dx^2} + \frac{du}{dx} - Cu = 0,\ x \in [0,2], \\ u(0) = 2, \\ \frac{du}{dx}|_{x=0} = -1. \end{aligned} \tag{3.25}$$

C takes three values 2, 6, and 12 for the *low*, the *medium*, and the *high* levels, respectively. To solve this ODE system, let N_f be 1,000. Eq. (3.26) provides the loss function used in training the NNs with all the activation functions. An NN of nine hidden layers with 50 neurons in each layer is utilized for training.

$$
\begin{aligned}
J(\tilde{\mathbf{\Theta}}) =& \frac{1}{N_f}\sum_{i=1}^{N_f}\Big(\frac{d^2u_{\mathbf{\Theta}}}{dx^2}|_{x_i} + \frac{du_{\mathbf{\Theta}}}{dx}|_{x_i} - Cu_{\mathbf{\Theta}}(x_i)\Big)^2 \\
&+\Big(u_{\mathbf{\Theta}}(0) - 2\Big)^2 \\
&+\Big(\frac{du_{\mathbf{\Theta}}}{dx}|_{x=0} - (-1)\Big)^2
\end{aligned}
\tag{3.26}
$$

The training loss plots in Figure 3.7 show a distinctly better convergence of the proposed activation function. As expected, a representative prediction on a set of 100 evenly spaced points in Figure 3.8 shows the almost perfect approximation (at the *high* level) of the exact function by the NNs with the Stan activation function. Table 3.1 displays the corresponding RE values for all the benchmark activation functions and Figure 3.9 shows a bar chart on the RE values.

3.2.3 Adaptive Hyperparameters

This section adopts some recent developments in the PINN literature to the proposed activation function. Particularly, the works of Wang et al.[143] and McClenny et al.[91] can be easily extended to the proposed Stan function. Wang et al.[143] proposed a learning rate annealing procedure (Adaptive Learning Rate - ALR) whereas McClenny et al.[91] proposed a weighting scheme (Adaptive Weighting - AW) for the points of the sets $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$. The default activation function is the tanh in both of these works.

The results in Figure 3.10 and Table 3.2 shows the adoption of ALR and AW for numerical studies on first-order and second-order ODE at the *high* level averaged over ten initializations. It is easy to see that both methods fail to significantly improve the tanh activation function. The significantly higher RE for the Adaptive Weighting scheme[91] (with Stan activation

function) might originate from alternating L-BFGS optimization for the NN parameters with Adam optimization for the adaptive parameters.

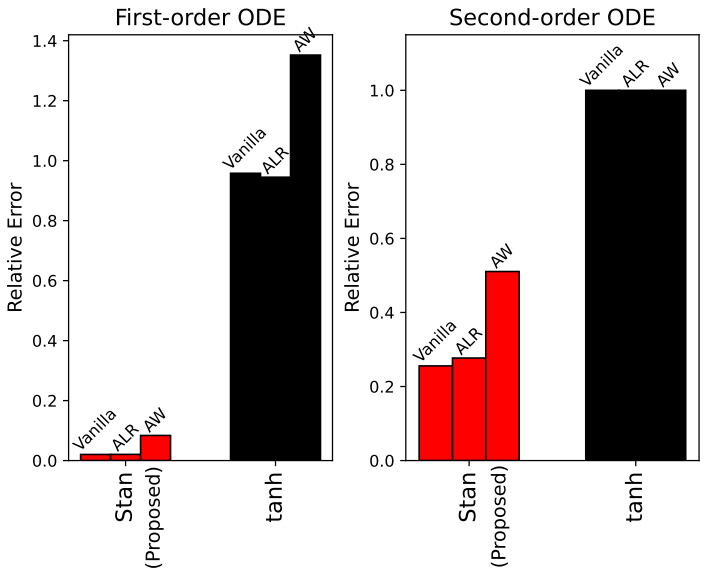


Figure 3.10: Comparison of the tanh and the proposed activation function on the study on adaptive hyperparameter in terms of relative error in prediction at the *high* level of first-order and second-order ODEs. "Vanilla" denotes PINN without any adaptive modifications.

Table 3.2: Test performance on adopting adaptive hyperparameters for the *high* level of first-order second-order ODE problems in terms of RE. "Vanilla" denotes PINN without any adaptive modifications.

	First-order ODE			Second-order ODE		
	Vanilla	ALR	AW	Vanilla	ALR	AW
Stan (Proposed)	**0.02**	**0.02**	**0.08**	**0.26**	**0.28**	**0.51**
tanh	0.96	0.94	1.35	1.00	1.00	1.00

3.3 Case Studies

Solving more elaborate differential equations that model complex physical systems further show the utility of the proposed Stan function. There are two types of problems considered

Table 3.3: Average performance in predicting the entire solution for the case studies on Klein-Gordon equation, Non-linear Heat Transfer in a Thin Plate, and Mass-Spring-Damper system in terms of RE.

	Klein-Gordon Equation			Non-linear Heat Transfer			Mass-Spring-Damper		
	L	M	H	L	M	H	L	M	H
Stan (Proposed)	**0.01**	**0.01**	**0.01**	**0.01**	**0.04**	**0.12**	**0.00**	**0.00**	**0.00**
tanh	0.02	0.16	0.04	0.03	0.18	0.39	0.13	0.40	0.44
tanh (AF) [60, 61]	0.02	0.01	0.06	0.03	0.14	0.35	0.18	0.42	0.52
Swish [114]	0.04	0.83	0.50	0.02	0.06	0.14	0.02	0.00	0.00
Rowdy [63]	0.07	0.04	0.04	0.04	0.17	0.29	0.04	0.01	0.00

in this section: (1) forward problems in Section 3.3.1 and Section 3.3.2, where the goal is to identify the solution of given a differential equation with its corresponding initial/boundary conditions similar to Section 3.2.2, and (2) inverse problem in Section 3.3.3, where the goal is to discover the coefficients of the differential equation given the solution at a selected number of points. Ten repetitions with different initializations of NN parameters and sets $\mathcal{F}, \mathcal{U}_1$, and $\mathcal{U}_2$ bring about the reported average performance metrics. Similar to the numerical studies, all the case studies are solved at three levels. The problem definition and results are provided in the main text. Appendix B.7 provides the loss function expressions and implementation details for all the case studies. Appendix B.8 and Appendix B.9 includes several additional case studies.

3.3.1 Klein-Gordon Equation

The time-dependent Klein-Gordon equation has several applications in solid-state physics and quantum physics[58]. The one-dimensional version of this equation is a benchmark problem in PINN[61, 143]. Consider the version of the problem provided by Wang et al.[143]. The governing differential equation system is as follows.

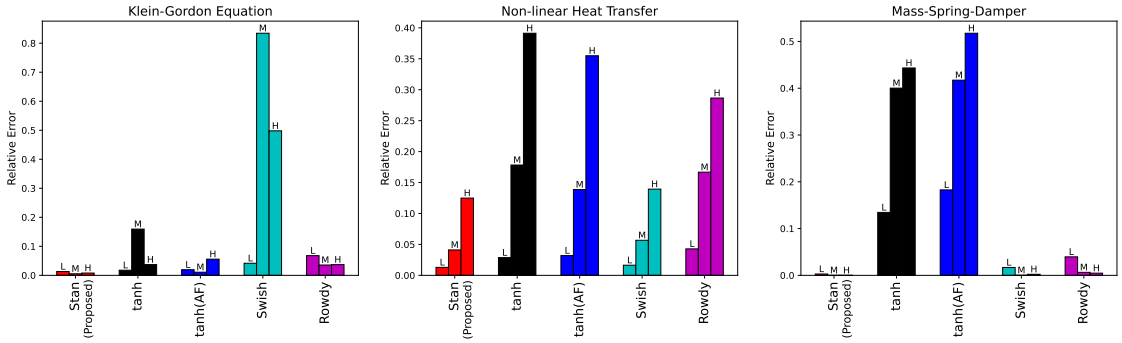


Figure 3.11: Comparison of activation functions on Klein-Gordon Equation, Non-linear Heat Transfer in a thinplate, and system identification of Mass-Spring-Damper in terms of relative error in predicting/reconstructing the solution. 'L' denotes *low*, 'M' denotes *medium*, and 'H' denotes *high* as described in the main text for corresponding problems. The proposed activation function is more robust than other activation functions to the magnitude of the output. Table 3.3 provides the values.

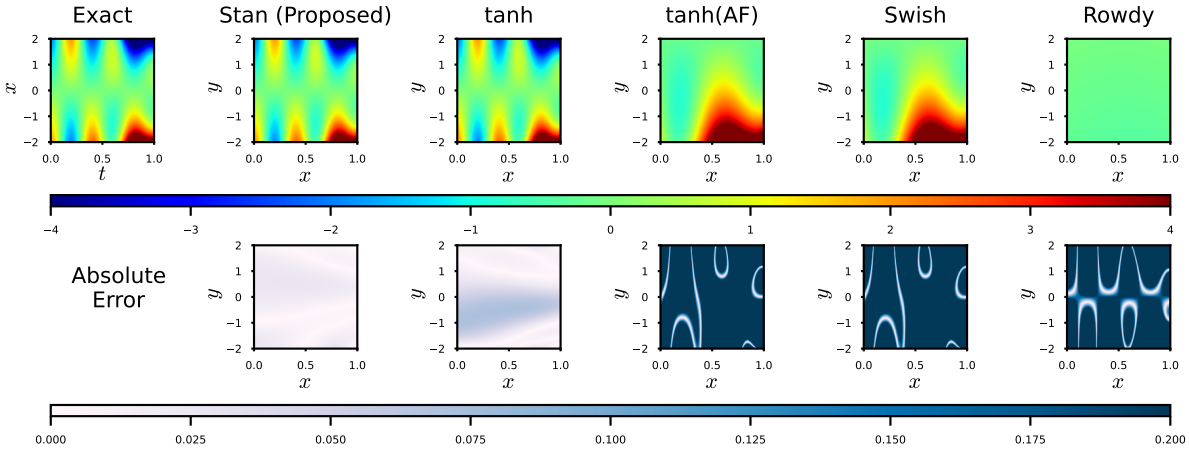


Figure 3.12: A representative prediction and corresponding error contours for solving the Klein-Gordon equation at the *high* level. The tanh function performs as good as other activation functions in this problem, whereas the proposed activation function is much better, as summarized in Table 3.3.

$$\begin{aligned}\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} + u^3 = & \Big[-25\pi^2 x\cos(5\pi t) + 6x^3 t - 6xt^3 \\ & +(xcos(5\pi t) + (xt)^3)^3 \Big], \ \forall x \in \Omega, t \in [0,1] \\ & u(x,0) = 0, \frac{\partial u}{\partial t}|_{u(x,0)} = 0, \forall\ x \in \Omega \\ & u(x,t) = h(x,t)\ \forall\ (x,t) \in \partial\Omega \times [0,1]\end{aligned} \tag{3.27}$$

The function $h(x,t)$ can be determined from the closed form solution $u(x,t) = x\cos(5\pi t) + (xt)^3$ (refer to [143]). Choosing various Ω gives the three levels for the problem. The *low* level is the same as in Wang et al.'s[143] work, i.e., $\Omega = [0,1]$. Let $\Omega = [0,2]$ be the *medium* level and $\Omega = [-2,2]$ be the *high* level.

Figure 3.12 shows the prediction and error contours at representative repetition for all the activation functions at the *high* level. Table 3.3 and Figure 3.11 convey the consistent performance of the proposed activation function across the levels. Note that the results shown are an average of 10 repetitions. Additionally, supplementing with ALR[143], the Stan function can achieve an average RE of 1.5e-3 at the *low* level, which is lower than the best value reported by Wang et al.[143].

3.3.2 Non-Linear Heat Transfer

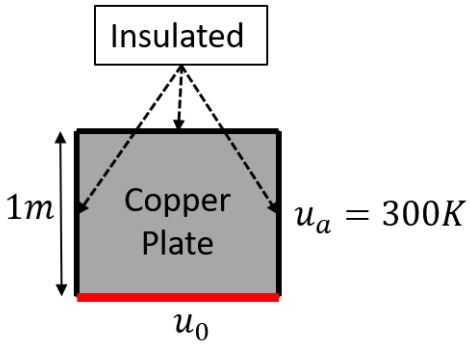


Figure 3.13: Problem setting for finding the transient temperature in the thin copper plate by applying a constant temperature of u_0 at the bottom of the plate.

Heat transfer in a thin plate is a well-studied problem with a wide range of engineering applications ranging from cooling computer systems to identifying the thermal history of an additively manufactured part. The characteristics of a simple version of the problem include a constant temperature applied to a square plate with relatively small thickness. Particularly, Figure 3.13 shows a copper plate with dimensions $1m \times 1m \times 1cm$ with a constant temperature

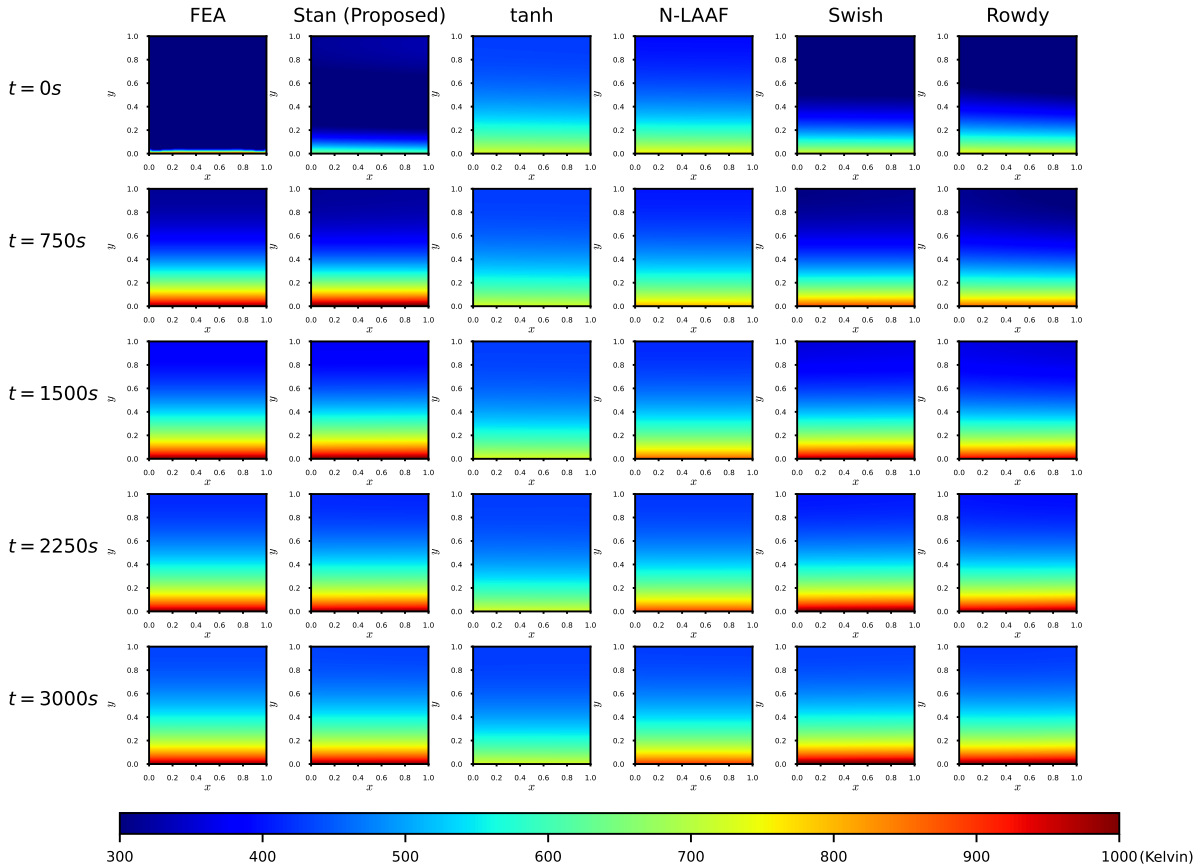


Figure 3.14: Comparison of the FEA solution of thermal history in a thin plate with PINN solutions of various activation functions at few representative time steps for the *medium* level. Except at $t = 0s$, which is the initial condition, the thermal history of FEA and the proposed Stan activation function look similar.

u_0 maintained at the bottom of the plate. Three values of u_0, 500K, 1000K, and 2000K, define the three levels *low*, *medium*, and *high*, respectively, for this problem. The ambient temperature is the room temperature, at 300K. The other three sides of the square insulate the plate. Convection and radiation in the top and bottom surfaces give away the heat to the surroundings, and conduction transfers the heat within the plate. A nonlinear PDE system in Eq. (3.28) models the transient temperature $u(x, y, t)$ in the plate.

$$\begin{aligned} \texttt{PDE}: \ \rho C_p t_c \frac{\partial u}{\partial t} - k t_c \nabla^2 u + 2h_c(u - u_a) \\ + 2\epsilon\sigma(u^4 - u_a^4) = 0, \ \{x, y, t\} \in \Omega \end{aligned} \tag{3.28}$$

$$\begin{aligned}
\texttt{DBC} &: u(x,0,t) = u_0,\ \{x,t\} \in \partial\Omega_{DBC} \\
\texttt{NBC1} &: \frac{\partial u}{\partial x}|_{(0,y,t)} = \frac{\partial u}{\partial x}|_{(1,y,t)} = 0,\ \{y,t\} \in \partial\Omega_{NBC1} \\
\texttt{NBC2} &: \frac{\partial u}{\partial y}|_{(x,1,t)} = 0,\ \{x,t\} \in \partial\Omega_{NBC2}.
\end{aligned} \tag{3.29}$$

In Eq. (3.28), ρ, C_p, and k are the density, the specific heat, and the conductivity of copper, respectively. Modeling the thermal history of the plate assumes all these material properties as constant irrespective of temperature. t_c is the thickness of the plate ($1cm$), u_a is the ambient temperature (300K), h_c is the convection coefficient, ϵ is the emissivity, and σ is the Stefan-Boltzmann constant. Appendix B.7 provides the values of these constants. Assume that the heat transfer process starts at $t = 0$, when the whole plate is at 300K.

Finite Element Analysis (FEA) using MATLAB R2022a[90] simulated the case, providing the approximation of the unknown solution. FEA assumed a mesh of 497 points at 101 times steps from 0 to 3000s to obtain the discrete solution. Note that PINNs provide (pseudo) closed-form solutions to the problem without discretization. The sample points chosen for PINN (for the sets $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$) are flexible, unlike the curated nodes in an FEA. Also, the time complexity of FEM increases rapidly with the number of nodes, whereas this is not the case with sampling points of PINN.

Figure 3.14 shows the solution in comparison with the PINNs' solutions in various time steps (for a representative initializing repetition) at the *high* level. PINN solution from the Stan activation function is much closer to the solution from FEA. Table 3.3 recorded the value of RE at the mesh points combining all the time steps, considering the FEA solution as the **ground truth**. The proposed Stan activation has the least error values in comparison with the other activation functions. The RE values are high, in comparison with other case studies in Figure 3.11, since the numerical solution dictates the reported values in this case.

3.3.3 System Identification of Mass-Spring-Damper

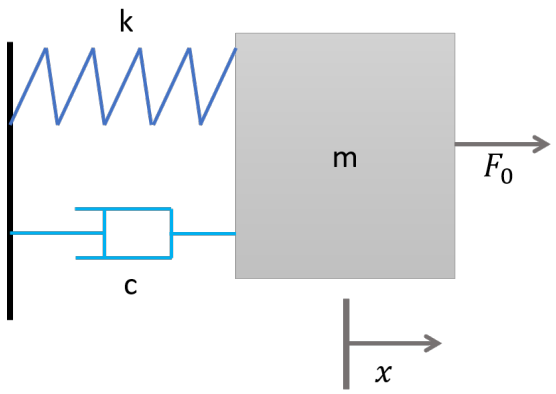


Figure 3.15: Problem setting for identifying the mass, spring constant and the damping constant for the known applied force.

Table 3.4: Average predictions (over 10 repetitions) for estimating the parameters in a Mass-Spring-Damper system. The proposed activation function can predict the system parameters consistently well.

	Low			*Medium*			*High*		
Actual Values	5.00	1.00	1.00	5.00	1.00	0.05	5.00	1.00	0.01
Average Estimates	$\hat{m}$	$\hat{c}$	$\hat{k}$	$\hat{m}$	$\hat{c}$	$\hat{k}$	$\hat{m}$	$\hat{c}$	$\hat{k}$
Stan (Proposed)	**4.99**	**1.00**	**1.00**	**5.00**	**1.00**	**0.05**	**5.00**	**1.00**	**0.01**
tanh	2.22	0.97	1.00	0.00	-0.01	0.07	0.00	0.05	0.04
tanh (AF) [60, 61]	0.55	0.59	1.00	0.00	-0.01	0.07	0.00	0.05	0.04
Swish [114]	4.49	0.90	1.00	4.99	1.00	0.05	3.52	1.06	0.01
Rowdy [63]	3.12	1.30	0.99	0.06	0.66	0.05	0.03	0.42	0.02

Mass-Spring-Damper (MSD) system is one of the classical and well-studied models in the field of mechanical engineering[93]. It has several applications ranging from vehicle dynamics to simulating human tendons[14]. A typical MSD system (in one-dimension) is modeled by

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = F_0, \tag{3.30}$$

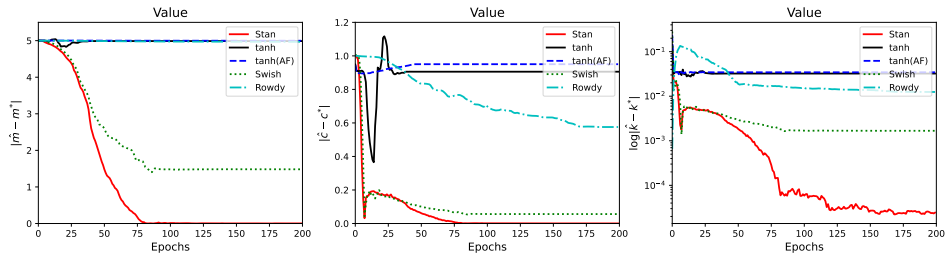


Figure 3.16: Average (of 10 repetitions) over training epochs of error in predicting the parameters m, c, and k at the *high* level with different activation functions for the system identification of a Mass-Spring-Damper. Stan function converges better than the benchmark activation functions.

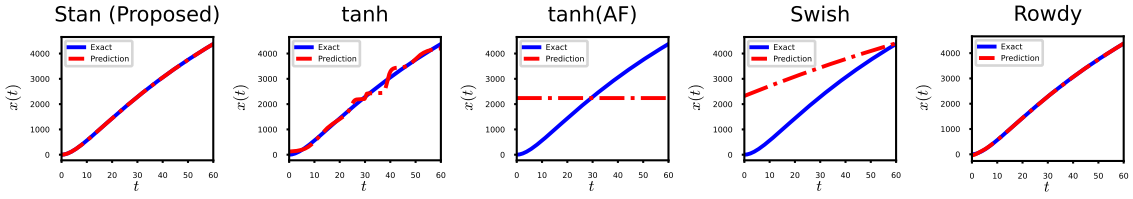


Figure 3.17: A representative output of PINN in learning the data along with the physics-information for Mass-Spring-Damper system at the *high* level for various activation functions.

where x is the displacement from the equilibrium position, t is the time, F_0 is the external force on the system, m is the mass, c is the damping coefficient, and k is the spring constant. A typical problem is to monitor the system response, i.e., x at several discrete steps of t a known force F_0 to estimate the system parameters (m, c, and k). Assume that the object in consideration is initially in rest at equilibrium (i.e., $x(0) = 0$ and $\frac{dx}{dt}|_{(t=0)} = 0$). Consider applying a constant known external force (F_0) of 100 Newton from time $t = 0$ as shown in Figure 3.15. The objective is to obtain m, c, and k given measured x at every $0.1s$ from 0 to $60s$.

Similar to the forward problems, the PDE, the boundary conditions, and the data make up the loss function[112], where the system parameters (m, c, and k) are trainable along with the NN parameters. Particularly, Eq. (3.31) gives the loss function.

To test this case, the actual solutions of the MSD system are simulated in Python *SciPy*-

odeint[142] at three levels. Exact k values (in N/m units) of 0.01, 0.05, and 1 are the *high*, the *medium*, and the *low* levels, respectively for this inverse problem. Exact m value is $5Kg$, and exact c value is $1N.s/m$.

$$
\begin{aligned}
J(\tilde{\mathbf{\Theta}}, \hat{m}, \hat{c}, \hat{k}) = \frac{1}{10000} \sum_{i=1}^{10000} \Bigg| & \hat{m} \frac{d^2 x_{\tilde{\mathbf{\Theta}}}}{dt^2}|_{t_i} + \hat{c} \frac{dx_{\tilde{\mathbf{\Theta}}}}{dt}|_{t_i} \\
& + \hat{k} x_{\tilde{\mathbf{\Theta}}}(t_i) - 100 \Bigg|^2 \\
& + \frac{1}{600} \sum_{i=1}^{600} |x_{\tilde{\mathbf{\Theta}}}(t_i) - x(t_i)|^2 \\
& + \left| \frac{dx_{\tilde{\mathbf{\Theta}}}}{dt}|_{t=0} \right|^2 .
\end{aligned}
\tag{3.31}
$$

Figure 3.16 shows the convergence of the system parameters to the exact value for the proposed activation function at the *high* level. The Swish activation function is the closest in performance to the proposed activation function at that level. Table 3.4 shows the predicted system parameters (on average over 10 repetitions) for all the activation functions at all the three levels. Further, Figure 3.17 shows the reconstruction of PINN with provided data and physics-information at *high* level at a representative repetition. Table 3.3 and Figure 3.11 show the average RE values in reconstructing the solution. Overall, Stan function performs significantly better than the other benchmark activation functions in this inverse problem. For the tanh function, attempts on using standardized data for identifying the parameters are not fruitful either.

3.4 Discussion

Based on the numerical studies in Section 3.2 and case studies in Section 3.3, the proposed Stan activation function can be significantly advantageous for the PINNs. The expressivity of derivatives discussed in Section 3.1.2 also has an effect on the initial loss value in first-order ODE. As expected, the first-order ODE's initial loss value (Figure 3.7) for the proposed activation function is significantly lesser than its counterparts.

The widely used tanh function and the adaptive modification are not very suitable for PINN, and are particularly worse when the solution is of higher-order magnitude. The Swish activation can potentially produce better results than the tanh in certain cases, since it has the sigmoid function multiplied with the input to the function, which can help in scaling the output to higher orders of magnitude. In general, the Rowdy activation function's results are competent with the proposed Stan activation function for PINNs. The goodness of the Rowdy is mainly due to the several sine terms in addition to the traditional tanh function. The nature of the Rowdy function addresses the spectral-bias[63] and can also tackle the output scaling better than the tanh, but the function is too complicated to practically use. The Rowdy function has too many parameters, which make it hard to tune, whereas the proposed Stan activation function has fewer parameters to tune (Appendix B.5). Indeed, to qualify PINN as a solver like the FEM, the convergence should not involve excessively searching for the hyperparameters. Also, the Rowdy activation function takes longer to train due to the computation of typically 3 to 9 terms in the activation function. The Stan function, in contrast, is very similar to the Swish activation function and the adaptive activation function[60] in terms of computation requirement.

Though "vanilla" PINNs are utilized throughout the work, any variant of PINN (e.g.,[59, 62, 100]) can benefit from utilizing proposed activation function. Similarly, the works on adap-

tive learning rate[143] and SA-PINN[91] can enhance the convergence even for the proposed activation function as shown in Section 3.2.3 and Section 3.3.1. This line of work will be explored more in the future.

3.5 Summary

This work proposed a Self-scalable tanh (Stan) for PINN, which contains a trainable parameter. Gradient descent algorithms can optimize the trainable parameter along with NN weights and biases. Our proposed Stan performs well on all types of case studies, comprising a regression problem, a range of forward problems, and an inverse problem. It is remarkably superior when the scale of the solution is potentially higher orders in magnitude. It can achieve better convergence in training and can provide better solutions compared to state-of-the-art activation functions. It accurately identified the system parameters of a Mass-Spring-Damper system with the simulated positional data in the inverse problem.

Though Stan improved the PINN to solve a wide range of problems, some aspects of Stan deserve further investigation. First, the initialization of the trainable parameters β_k^i's needs additional research. This work initialized the β_k^i's for all the neurons with constant values (or with a small standard deviation), and most of those values turned out to be ones after tuning (Appendix E). The second aspect which needs attention is the loss landscape of the PINNs. The Stan function creates a suitable loss landscape (refer to Section 3.1.3), but the nature of the loss function constructed by any particular form of differential equation is still not well characterized. A better understanding of the loss functions can lead to better ways to train the PINN to solve complex problems.

Chapter 4

Fast Simulations of Additive Manufacturing with Machine Learning

The chapter is organized as follows. Section 4.1 provides the theoretical background for modeling the thermal distribution in L-PBF. Section 4.2 describes the two-stage implementation in a two-dimensional plate and the FNO method for predicting highly non-stationary heat input. Section 4.3 provides the summary and future work.

4.1 Background

The background necessary for this work is of two parts. The first aspect in Section 4.1.1 details the thermal model. The thermal model describes the thermal distribution of a part subjected to heat input through a source. Though it is natural to extend, for simplicity, this work ignores the fine-scale melting involved in typical metal AM processes like L-PBF. The second aspect in Section 4.1.3 details FNO, the data-driven paradigm used in this work.

4.1.1 Thermal Model of Metal AM

A generic heat transfer model, in its simplest form, is given as

$$\rho c_p \frac{\partial T}{\partial t} = -\vec{\nabla}.\vec{q} + Q \tag{4.1}$$

where the temperature is T, time is t, $\vec{\nabla}.$ is the divergence operator, $\vec{q}$ is heat flux, and Q is the internal heat source. The density of the material is given by ρ, and the specific heat is given by c_p. The heat equation can be solved numerically with appropriate initial and boundary conditions. Considering a solid medium, the heat flux through the manufactured part is given as

$$\vec{q} = -\kappa \vec{\nabla} T \tag{4.2}$$

where κ gives the conductivity of the medium. Note that the material properties (ρ, c_p, or κ) can be a function of temperature. The heat source Q models the heat input to the part if it is a volumetric. Considering a heat flux input for the laser source is also an acceptable model [47]. In most AM processes that use a laser-based heat source, a Gaussian ellipsoidal distribution is used as the heat input from the laser [45, 47]. As a 2D heat flux laser input on a 3D part,

$$\vec{q_l} = \frac{3P\eta}{\pi ac} \exp\left(-\frac{3x^2}{a^2} - \frac{3(z+v_s t)^2}{c^2}\right)\hat{q} \tag{4.3}$$

on the surface with unit normal $\hat{q}$ that interacts with the laser. P denotes laser power, η denotes the laser absorptivity of the material, v_s denotes the scanning speed (the speed with which the laser moves), and a and c are the axis dimensions of the ellipsoid. z coordinate denotes the concerned point parallel to the motion of the heat source, and x coordinate

denotes the point in the direction perpendicular to both the motion of the heat source and $\hat{q}$. As a 3D volumetric heat source,

$$Q = \frac{6\sqrt{3}P\eta}{\pi abc} \exp\left(-\frac{3x^2}{a^2} - \frac{3y^2}{b^2} - \frac{3(z+v_s t)^2}{c^2}\right) \tag{4.4}$$

where, apart from the variables used in Eq. (4.3), y coordinate denotes the point in the direction of depth of the material and c denotes the corresponding length of the ellipsoid axis. For a L-PBF process, the initial temperature is typically room temperature (considered as 300K). The space surrounding the build is filled with metal powder that dissipates the heat. Also, convection and radiation dissipate the heat away from the build. The heat losses are typically negligible and optional to include in the model.

4.1.2 Physics-informed Neural Networks for L-PBF

PINNs solve differential equations in a non-traditional way. Let $\mathbf{x} \in \Omega \subset \mathbb{R}^d$ denote the spatio-temporal location and $T(\mathbf{x}) \in \mathbb{R}\ \forall\ \mathbf{x} \in \Omega$ denote the solution to be determined from the PDE

$$\begin{aligned} \rho c_p \frac{\partial T(\mathbf{x})}{\partial t} - \kappa \nabla^2 T(\mathbf{x}) = 0,\ \mathbf{x} \in \Omega, \\ T(\mathbf{x}) = g_1(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_1, \\ \nabla T(\mathbf{x}) = g_2(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_2; \end{aligned} \tag{4.5}$$

where ∇ denotes the gradient operator, g_1, and g_2 are known functions, and T is the unknown solution to be learned. $T(\mathbf{x}) = g_1(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_1$ and $\nabla T(\mathbf{x}) = g_2(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_2$ are the Dirichlet and Neumann boundary conditions (DBC and NBC), respectively. For the purpose of PINN, the initial condition is treated exactly like a DBC.

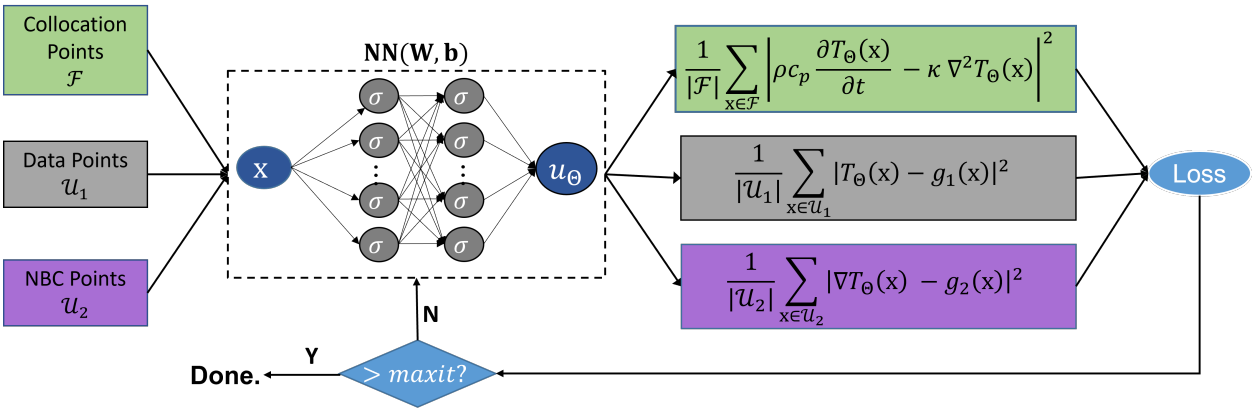


Figure 4.1: Schematic of PINN for a general differential equation system. The training of the PINN starts with the sets of points $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$, which are subsets of Ω,$\partial\Omega_1$, and $\partial\Omega_2$, respectively. The points in set $\mathcal{F}$ form the first part of the loss function, the points in set $\mathcal{U}_1$ form the second part, and the points in set $\mathcal{U}_2$ form the third part, as determined by the NN at current training step. The loss is back-propagated to train the parameters of the NN.

Consider a fully-connected NN of depth $D+1$ corresponding to a network with d-dimensional inputs (4-dimensional (x,y,z,t) for a real case), D hidden layers, and an output layer. Assume all the hidden layers have P neurons each. $\mathbf{x} \in \Omega$ is the input to the NN, and the NN approximation of the solution function $T(\mathbf{x})$ is the output. The fully connected NN can be represented as follows for a given input $\mathbf{x}$.

$$\begin{aligned} \mathbf{L}_1 &= \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \\ \mathbf{z}_k &= \sigma(\mathbf{L}_k) \ \forall \ k = \{1, 2, ..., D\} \\ \mathbf{L}_k &= \mathbf{W}_k\mathbf{z}_{k-1} + \mathbf{b}_k \ \forall \ k = \{2, 3, ..., D\} \\ T_{\mathbf{\Theta}}(\mathbf{x}) &= \mathbf{W}_{D+1}\mathbf{z}_D + \mathbf{b}_{D+1} \end{aligned} \tag{4.6}$$

where $\mathbf{W}_k \ \forall \ k \in \{1, 2, ..., D+1\}$ represent the weight matrices and $\mathbf{b}_k \ \forall \ k \in \{1, 2, ..., D+1\}$ represent the bias vectors of corresponding layers in the NN. Note that $\mathbf{W}_k \in \mathbb{R}^{P\times P} \ \forall \ k \in \{2, ..., D\}$, $\mathbf{b}_k \in \mathbb{R}^{P\times 1} \ \forall \ k \in \{1, 2, ..., D\}$, $\mathbf{W}_1 \in \mathbb{R}^{P\times d}$, and for a scalar output $T(\mathbf{x})$, $\mathbf{W}_{D+1} \in \mathbb{R}^{1\times P}$ and $\mathbf{b}_{D+1} \in \mathbb{R}^{1\times 1}$. $\mathbf{z}_k \in \mathbb{R}^{P\times 1} \ \forall \ k \in \{1, 2, ..., D\}$ is the output vector of k^{th} hidden

layer of the network and input to the next layer. $\sigma(\cdot)$ is the activation function (Stan [43] is used throughout this work) that acts element-wise on vector $\mathbf{L}_k \in \mathbb{R}^{P\times 1}\ \forall\ k \in \{1, 2, ..., D\}$. $\mathbf{\Theta}$ collectively denotes all the trainable weights and biases.

Learning the solution to Eq. (4.5) implies $T_{\mathbf{\Theta}^*}(\mathbf{x}) \approx T(\mathbf{x})$, where $\mathbf{\Theta}^*$ is the optimal value of $\mathbf{\Theta}$. Choosing three finite sets of points, $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$, such that $\mathcal{F} \subset \Omega$, $\mathcal{U}_1 \subset \partial\Omega_1$, and $\mathcal{U}_2 \subset \partial\Omega_2$ aids in learning the $\mathbf{\Theta}^*$. These sets define the loss function as

$$J(\mathbf{\Theta}) = w_{\mathcal{F}}\text{MSE}_{\mathcal{F}} + w_{\mathcal{U}_1}\text{MSE}_{\mathcal{U}_1} + w_{\mathcal{U}_2}\text{MSE}_{\mathcal{U}_2}, \tag{4.7}$$

where

$$\begin{aligned}
\text{MSE}_{\mathcal{F}} &= \frac{1}{|\mathcal{F}|}\sum_{\mathbf{x}\in\mathcal{F}} |\rho c_p \frac{\partial T_{\mathbf{\Theta}}(\mathbf{x})}{\partial t} - \kappa\nabla^2 T_{\mathbf{\Theta}}(\mathbf{x})|^2, \\
\text{MSE}_{\mathcal{U}_1} &= \frac{1}{|\mathcal{U}_1|}\sum_{\mathbf{x}\in\mathcal{U}_1} |T_{\mathbf{\Theta}}(\mathbf{x}) - g_1(\mathbf{x})|^2, \text{and} \\
\text{MSE}_{\mathcal{U}_2} &= \frac{1}{|\mathcal{T}_2|}\sum_{\mathbf{x}\in\mathcal{U}_2} |\nabla u_{\mathbf{\Theta}}(\mathbf{x}) - g_2(\mathbf{x})|^2,
\end{aligned}$$

and $w_{\mathcal{F}}$, $w_{\mathcal{U}_1}$, and $w_{\mathcal{U}_2}$ are the corresponding weights for each segment of the loss function. The points in set $\mathcal{F}$ are the PDE/ODE residual points or the collocation points since the residual of the PDE/ODE, i.e., $\rho c_p \frac{\partial T_{\mathbf{\Theta}}(\mathbf{x})}{\partial t} - \kappa\nabla^2 T_{\mathbf{\Theta}}(\mathbf{x})$ uses these points for calculation. Likewise, $\text{MSE}_{\mathcal{F}}$ is the `PDE/ODE residual loss`. $\text{MSE}_{\mathcal{U}_1}$ is the `Dirichlet Boundary loss` or simply `Data loss`, since these are the only directly available data in the form of inputs and outputs to train the PINN. Also, note that any available data adds to the `Data loss` part of the loss function, relaxing the condition that this part should be from the boundary ($\partial\Omega_1$). This relaxation makes the PINN more flexible than the FEM and capable of solving inverse problems. The third part of the loss function $\text{MSE}_{\mathcal{U}_2}$ is the `Neumann Boundary loss`.

The optimal set of parameters $\mathbf{\Theta}^*$ minimizes the loss function in Eq. (4.7). To solve the

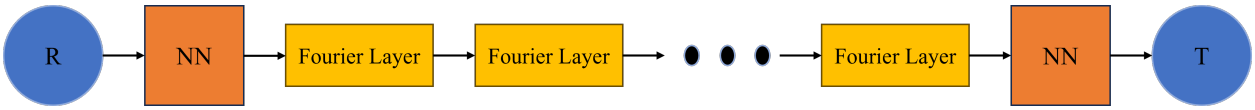


Figure 4.2: The process parameters (R) are the input to the FNO. The input is connected to the Fourier layers via a projection NN that lifts it to the higher dimensional space. The Fourier layers are again projected back to the target dimension via another NN. The target is the thermal distribution. The architectures used in this work typically use four Fourier layers.

system in Eq. (4.5), all the chosen points in the corresponding sets $\mathcal{F}$, $\mathcal{U}_1$, and $\mathcal{U}_2$ should satisfy the corresponding conditions in Eq. (4.5), and thus should minimize the mean-squared errors. Hence, the training to solve the PDE/ODE system involves finding $\boldsymbol{\Theta}^*$ where

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}). \tag{4.8}$$

One of the forms of gradient descent algorithm, Adam[68] or L-BFGS[19] approximates the solution to the problem in Eq. (4.8) iteratively. Figure 4.1 shows the schematic of training PINN for given process parameters.

4.1.3 Fourier Neural Operator

Operator Learning

This work proposes to use Fourier Neural Operators (FNO) [78] to learn the solutions of the heat equations in AM. Firstly, by definition, operator learning [69] learns the mapping between infinite-dimensional function spaces. Neural Operators (NO) [69] specifically is a new class of methods that extends the existing DL to functional spaces. FNOs use Fourier layers instead of convolutional layers in well-known Convolutional Neural Networks (CNNs) [77]. These Fourier layers better represent functional spaces than a typical discretization in conventional neural networks [78]. The operator learning methods [69] have two significant

advantages over conventional DL methods.

1. **Learning in a continuous domain:** Most developments in the field of DL have focused on signals that humans can directly recognize in the form of images, videos, or texts. These methods are limited to learning discrete spaces like vectors, matrices, or tensors. Unlike these data, the simulation inputs and outputs are typically in the functional space, i.e., defined at every spatial-temporal location in the domain.

2. **Amount of data:** Most methods in the conventional DL assume the availability of a large amount of data to train the models. As mentioned in Section 1, one of the significant reasons for building alternate methods to simulations is to limit the time taken to generate data for different scenarios or process parameter settings. Thus, generating the data required to train the conventional DL models is implausible.

Apart from FNO, there are multiple operator learning networks in the recent literature, including DeepONets [87], Graph NO [79], and Low-rank NO [69]. Multiple studies show the superiority of NO-based methods over DeepONets [69, 80]. The goodness of the Fourier Transform-based NO, particularly in the context of heat equation, can be attributed to the inherent connection between the two.

Consider the heat equation in Eq (4.1) rewritten as

$$\rho c_p \frac{\partial T(\mathbf{x}, t)}{\partial t} + \vec{\nabla}.\vec{q}(\mathbf{x}, t) = Q(\mathbf{x}, t) \tag{4.9}$$

where $\mathbf{x}$ denotes the spatial location (in a generic n-dimensional space) and t denotes the time. As in Eq. (4.4), the process parameters define the characteristics of the function $Q(\mathbf{x}, t)$. We aim to solve the heat equation for a given process parameter setting to give the corresponding $T(\mathbf{x}, t)$. Let $\mathcal{P}$ denote the set of all the process parameters. The goal is to

build a model $\mathcal{G}_\theta$ parameterized by θ, that can be learned from data. Mathematically,

$$\mathcal{G}_\theta : \mathcal{P} \rightarrow T_{\mathcal{P}}(\mathbf{x}, t) \tag{4.10}$$

where G_θ is characterized by FNO. Note that physically $\mathcal{P}$ characterizes the entire laser movement, not just the variables in Eq. (4.4). Thus, the heat source term $Q_{\mathcal{P}}(\mathbf{x}, t)$ can be considered as the function that embeds all the process parameters. The material properties can also be considered a part of $\mathcal{P}$. Still, for this work, the material is assumed to be the same, irrespective of the process parameter setting. Thus, the modeling problem is to learn θ such that

$$\mathcal{G}_\theta : Q_{\mathcal{P}}(\mathbf{x}, t) \rightarrow T_{\mathcal{P}}(\mathbf{x}, t) \tag{4.11}$$

FNOs use the well-known Fast Fourier Transform (FFT) to build a neural network architecture in the Fourier space. Figure 4.2 shows an FNO architecture. In addition to a linear layer with a non-linear activation function, each Fourier layer transforms the input to a Fourier domain, filters the transformed input, and applies an inverse Fourier transform to retrieve the signal in the original space. Readers are referred to [78] for the details of the Fourier layer in Figure 4.2. Once the FNO is trained with sufficient data, evaluating thermal history $T_{\mathcal{P}}(\mathbf{x}, t)$ for any given process parameters $\mathcal{P}$ is a straightforward instantaneous evaluation from $Q_{\mathcal{P}}(\mathbf{x}, t)$.

4.2 Case Studies

Case studies are carried out for both stages (simulation with PINN and training and prediction with FNO). For simplicity, a 2-dimensional case is considered in Section 4.2.1 where

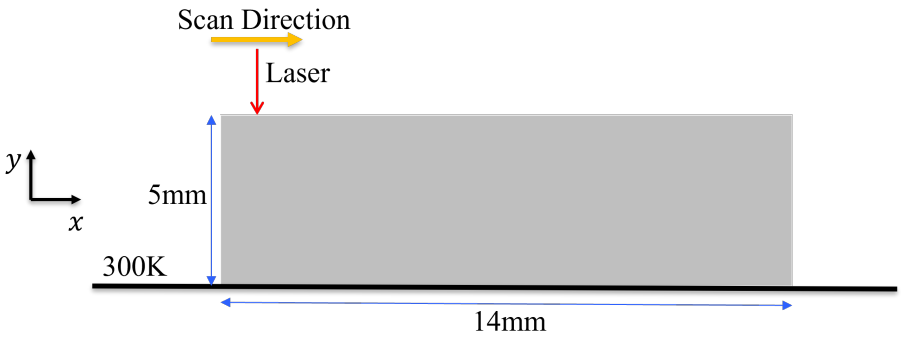


Figure 4.3: Problem setting for modeling heat transfer in 2-dimension.

the heat transfer model is solved via PINN to generate data. The generated data is used for training the FNO to compare it with the solution generated from FEM eventually. Initial simulation with PINN alone is provided in Section 4.2.2.

4.2.1 Heat Source at the Boundary

Our previous work [43] shows heat-transfer modeling with PINN. For simplicity, a common practice is to ignore the convection and radiation boundary conditions. Consider a 2-D plate with dimensions $14mm \times 5mm$ as shown in Figure 4.3. The laser is considered to be a point source that moves in x-direction from one end to the other. Specifically, the heat transfer

model in Eq. (4.5)

$$\begin{aligned}
\rho c_p \frac{\partial T(x,y,t)}{\partial t} - \kappa \nabla^2 T(x,y,t) &= 0, \ \forall\ x \in [0,14]; y \in [0,5]; t > 0, \\
T(x,0,t) &= 300, \ \forall\ x \in [0,14]; t > 0, \\
T(x,y,0) &= 300, \ \forall\ x \in [0,14]; y \in [0,5], \\
\frac{\partial T(x,y,t)}{\partial x}|_{x=14} = \frac{\partial T(x,y,t)}{\partial x}|_{x=0} &= 0, \ \forall\ y \in [0,5]; t > 0, \\
\frac{\partial T(x,y,t)}{\partial x}|_{y=5} &= Q(x,t); Q(x,t) = \frac{-2L_p\eta}{\pi r_l^2} \exp^{\frac{-2(x-x_l^2)}{r_l^2}} \ \forall\ x \in [0,14]; t > 0.
\end{aligned} \tag{4.12}$$

where L_p is the laser power, η is the laser absorptivity, r_l is the radius of the laser, and x_l denotes the position of the laser in time. For this particular setting, $x_l = 1 + L_s t$, where L_s is the scan speed of the laser. The material properties and the process parameters are considered constant in this setting.

PINN Simulation

For $L_p = 195W$ and $L_s = 80mm/s$, simulations were carried out with PINN and FEM. The implementation details are provided in Appendix C.2. Figure 4.4 shows the comparison between PINN and FEM simulations at a chosen laser power. The results are very consistent. The Root Mean Squared-Error (RMSE) at the FEM nodes between the PINN and FEM simulations is approximately **45K**.

FNO Training and Prediction

PINN, with transfer learning, is used for simulating the heat transfer with 100 different laser power values. 80 laser power values with the corresponding thermal history are used for

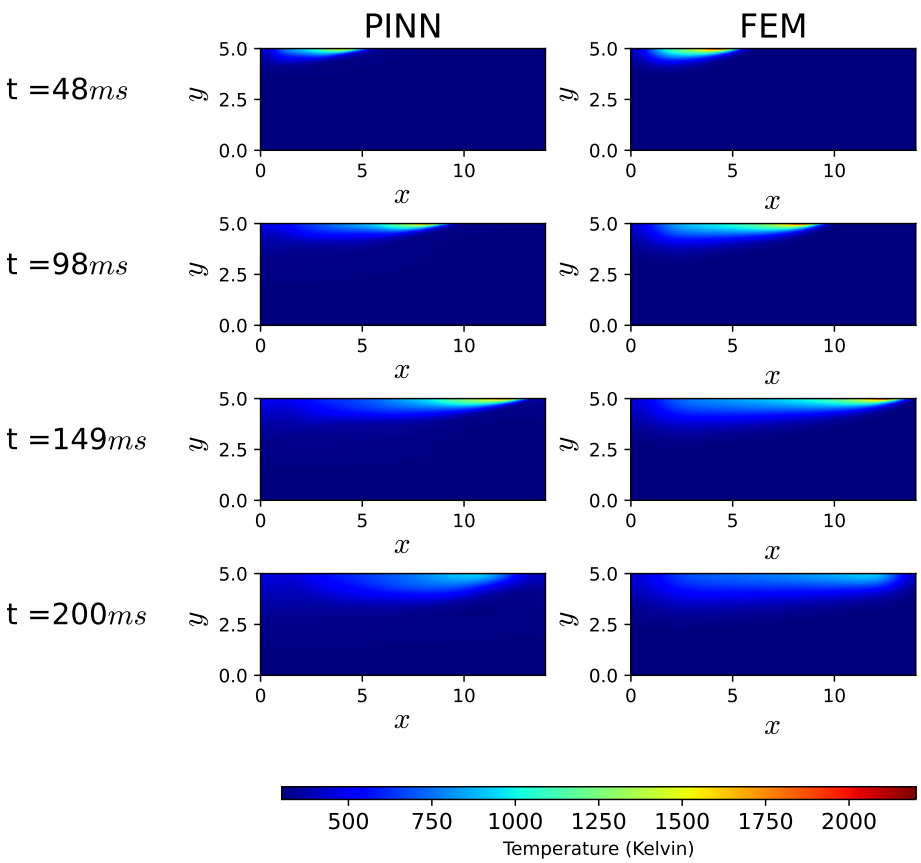


Figure 4.4: Comparing PINN and FEM prediction at 195W.

training the FNO. Figure 4.5 shows the FNO prediction along with PINN prediction and FEM prediction at a given laser power. The RMSE between the FNO prediction and PINN prediction is **13K**, whereas the RMSE between FNO prediction and FEM prediction is **85K** at 185W. Note that the PINN predictions are mesh-free, enabling high-resolution data to be trained on FNO.

4.2.2 Internal Heat Generation

This case study involves a more complex set of process parameters added directly to the heat source term of Eq. (4.1). In particular, the two-dimensional plate is assumed to be

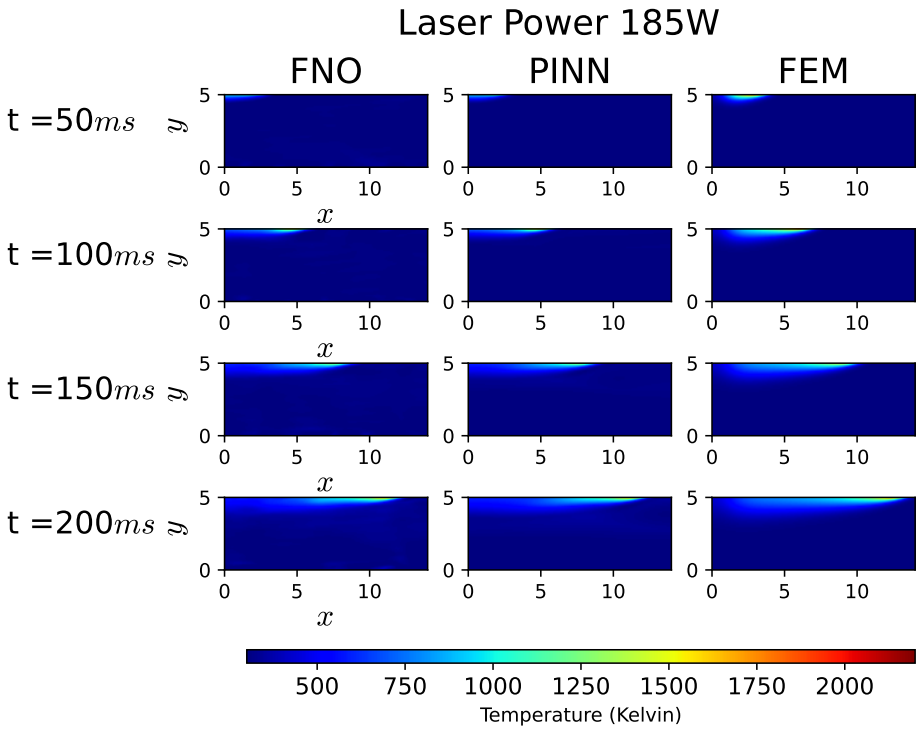


Figure 4.5: Comparing FNO trained on PINN data with PINN prediction itself and FEM prediction.

subject to multiple scans. In this case, the process parameters are laser power, scan speed, and number of scans. These three process parameters create a highly non-stationary thermal distribution in the plate. A sequential sampling of all process parameters resulted in 900 combinations with 10 levels of laser power, 10 levels of scan speed, and 9 different numbers of scans. 400 of these experiments were randomly chosen for training, and the rest were used to test the predicted model. In this case, the process parameters are encoded onto a matrix as 3 Gaussian distributions, as in Figure 4.6. Each circular Gaussian distribution denotes one of the process parameters, and the intensity in the image denotes the relative magnitude (for example, for laser power, brightness depends on the value 50 to 250). The FNO is compared with other well-known DL methods like UNet and ResNet, as shown in Figure 4.6. Though all the methods learn a roughly increasing temperature trend, even visually, FNO is

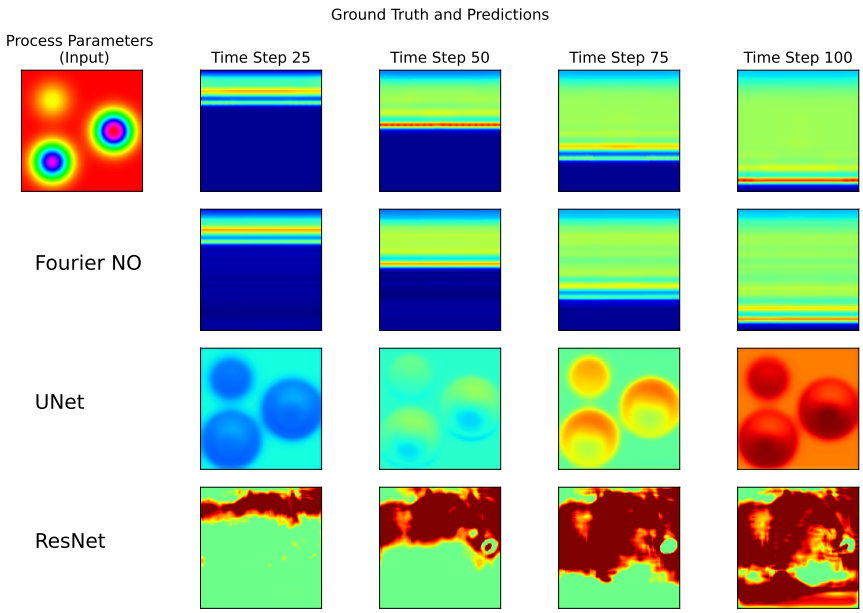


Figure 4.6: The comparison of temperature distribution predicted by different DL methods at $230W$ laser power, $1.1m/s$, and 10 scans.

far better than the established DL methods. In particular, UNet predicts a modification of the input pattern, whereas ResNet learns an overall pattern far from the ground truth. The comparison on the entire test set (500 sets of process parameters) is provided in Table 4.1.

Table 4.1: Comparison on DL methods for predicting thermal distribution in metal AM.

Algorithms	Testing Relative Error	Training Time (minutes)
FNO	0.0512	19.5
UNet	0.6437	29
ResNet	0.6056	26

4.3 Summary

The current results show promising directions of (1) using PINN as a simulator for L-PBF and (2) using FNO to learn the operator that directly works on input process parameters to predict the output thermal history. Using PINN is advantageous due to the mesh-free nature of the solution and its flexibility to include process data. Using FNO is advantageous

as we might require instantaneous predictions of thermal history for a new set of process parameters. The future work is to add more complex heat inputs to both PINN and FNO, and to simulate them for different geometries.

Chapter 5

Conclusions and Future Work

This dissertation implemented a Bayesian Optimization (BO) method DGP-SI-BO, Self-scalable Tanh (Stan) activation function for Physics-informed Neural Networks (PINNs), and Fourier Neural Operators (FNO) for enabling data analysis for metal Additive Manufacturing (AM). The BO method helps design the experiments towards optimizing a quality metric whereas the activation function enables simulations to produce multi-scale solutions. Lastly, the FNO has been shown to enable fast and accurate predictions on . The proposed BO method is better than benchmark BO methods for optimizing process parameters (particularly laser power, scan speed, and hatch spacing). Several theoretical and empirical justifications were provided for using the Stan activation function over the literature activation functions for PINNs. The applicability of the Stan activation function extends well beyond the application at hand, i.e., Laser Powder Bed Fusion (L-PBF). FNO provides adequate simulations (a few hundred typically) and can learn the high-dimensional mapping between the process parameter space and the solution of Partial Differential Equations (PDEs). FNO is particularly suited for PDE-based data compared to any other deep learning method.

In terms of application, the main conclusion drawn from this work is that accurate physics-based modeling is desirable. Currently, the physics of the AM processes remains vastly a mystery. Even with the known knowledge, solving the equations that capture multiscale physics to replicate the process faithfully is practically impossible. So, developing tools

for process optimization is as essential as developing tools for faster and more accurate simulations. Along those lines, this dissertation provides three significant tools for enabling Digital Twin in AM.

1. Sequential design of experiments is useful for process optimization, particularly when the physics of the process is unknown, and the mapping between process parameters and the quality output can be treated as a "black-box". This dissertation provides a Deep Gaussian Process (DGP)-based surrogate model for the Bayesian sequential design of experiments (Chapter 2).

2. PINNs are a new paradigm for solving PDEs, and they provide alternate ways to get a fast and accurate physics-based process simulation (Chapter 3). PINNs provide mesh-free solutions, enable combining the experimental data with PDE-based models, and solve the inverse problem of identifying the system parameters (typically the coefficients in the PDEs).

3. Instance-based solving of PDEs is cumbersome and time-consuming. FNO provides a neat Fourier transform-based model to predict the simulation output by building a data-driven mapping between the input (characterized by controllable process parameters) and output (high-dimensional thermal or stress tensor) functional spaces.(Chapter 4).

The future of DTs in AM revolves around combining different machine learning techniques to extend it to provide high-quality manufactured parts. In particular, three future directions are worthwhile to pursue.

1. Combining experimental and simulation to accelerate process optimization. This can be achieved by using the known-physics-based simulations as a Bayesian prior for

experimental data. The proposed DGP-SI-BO provides a convenient framework for the same. Case studies in Chapter 3 also show combining sensor data with PDEs.

2. Multiscale integration in physics-based simulations is necessary. For instance, current physics-based simulations approximate the melt pool level influence of process parameters on the part level quality. The main reason for the approximation is the hefty computation requirement. FNOs accelerate the simulations; thus, multilevel integration is an extension that can be explored.

3. Edge-device-friendly algorithms and models for process monitoring and control are essential. Though edge devices are getting more powerful, there are several limitations in using the developed models in edge devices that can be used for real-time process monitoring using sensors and control using industrial controllers. Model order reduction and federated learning using the proposed methods is also an important direction to pursue.

Bibliography

[1] Amir M Aboutaleb, Linkan Bian, Alaa Elwany, Nima Shamsaei, Scott M Thompson, and Gustavo Tapia. Accelerated process optimization for laser-based additive manufacturing by leveraging similar prior studies. *IISE Transactions*, 49(1):31–44, 2017.

[2] Parand Akbari, Francis Ogoke, Ning-Yu Kao, Kazem Meidani, Chun-Yu Yeh, William Lee, and Amir Barati Farimani. Meltpoolnet: Melt pool characteristic prediction in metal additive manufacturing using machine learning. *Additive Manufacturing*, 55: 102817, 2022.

[3] Ali Al-Safwan, Chao Song, and Umair bin Waheed. Is it time to swish? comparing activation functions in solving the helmholtz equation using physics-informed neural networks. *arXiv preprint arXiv:2110.07721*, 2021.

[4] WE Alphonso, M Baier, S Carmignato, JH Hattel, and M Bayat. On the possibility of doing reduced order, thermo-fluid modelling of laser powder bed fusion (l-pbf)–assessment of the importance of recoil pressure and surface tension. *Journal of Manufacturing Processes*, 94:564–577, 2023.

[5] Edward Anderson and Harrison Nguyen. When can we improve on sample average approximation for stochastic optimization? *Operations Research Letters*, 48(5):566–572, 2020.

[6] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021.

[7] Autodesk. Netfabb local simulation. *Manual*, 1:1–100, 2020.

[8] Zhaojun Bai, Patrick M Dewilde, and Roland W Freund. Reduced-order modeling. *Handbook of numerical analysis*, 13:825–895, 2005.

[9] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization, 2020.

[10] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[11] David A Belsley, Edwin Kuh, and Roy E Welsch. *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley & Sons, 2005.

[12] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

[13] Garrett Birkhoff. Mathematics for engineers—iii: Dimensional analysis of partial differential equations. *Electrical Engineering*, 67(12):1185–1188, 1948.

[14] Reinhard Blickhan. The spring-mass model for running and hopping. *Journal of biomechanics*, 22(11-12):1217–1227, 1989.

[15] George EP Box and Kenneth B Wilson. On the experimental attainment of optimum conditions. In *Breakthroughs in statistics*, pages 270–310. Springer, 1992.

[16] Eric Brochu, Nando De Freitas, and Abhijeet Ghosh. Active preference learning with discrete choice data. In *NIPS*, pages 409–416, 2007.

[17] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 1:1–49, 2010.

[18] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual review of fluid mechanics*, 52:477–508, 2020.

[19] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

[20] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6):0608011–15, 2021.

[21] Yunus A Cengel. *Introduction to thermodynamics and heat transfer: Engineering*. McGraw-Hill, 2008.

[22] Anindya Chatterjee. An introduction to the proper orthogonal decomposition. *Current science*, pages 808–817, 2000.

[23] Jiangce Chen, Wenzhuo Xu, Martha Baldwin, Björn Nijhuis, Ton van den Boogaard, Noelia Grande Gutiérrez, Sneha Prabha Narra, and Christopher McComb. Capturing local temperature evolution during additive manufacturing through fourier neural operators. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 87295, page V002T02A085. American Society of Mechanical Engineers, 2023.

[24] Lequn Chen, Xiling Yao, Kui Liu, Chaolin Tan, and Seung Ki Moon. Multisensor

fusion-based digital twin in additive manufacturing for in-situ quality monitoring and defect correction. *Proceedings of the Design Society*, 3:2755–2764, 2023.

[25] Francisco Chinesta, Pierre Ladeveze, and Elias Cueto. A short review on model order reduction based on proper generalized decomposition. *Archives of Computational Methods in Engineering*, 18(4):395–404, 2011.

[26] Lucas Santos Dalenogare, Guilherme Brittes Benitez, Néstor Fabián Ayala, and Alejandro Germán Frank. The expected contribution of industry 4.0 technologies for industrial performance. *International Journal of production economics*, 204:383–394, 2018.

[27] Andreas Damianou and Neil D Lawrence. Deep gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR, 2013.

[28] James R Deneault, Jorge Chang, Jay Myung, Daylond Hooper, Andrew Armstrong, Mark Pitt, and Benji Maruyama. Toward autonomous additive manufacturing: Bayesian optimization on a 3d printer. *MRS Bulletin*, 46(7):566–575, 2021.

[29] Jean Donea and Antonio Huerta. *Finite element methods for flow problems*. John Wiley & Sons, 2003.

[30] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[31] Matthew M Dunlop, Mark A Girolami, Andrew M Stuart, and Aretha L Teckentrup. How deep are deep gaussian processes? *Journal of Machine Learning Research*, 19 (54):1–46, 2018.

[32] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51:357–377, 2019.

[33] Vincent Dutordoir, Hugh Salimbeni, Eric Hambro, John McLeod, Felix Leibfried, Artem Artemev, Mark van der Wilk, James Hensman, Marc P Deisenroth, and ST John. Gpflux: A library for deep gaussian processes. *arXiv preprint arXiv:2104.05674*, 1(1):1–7, 2021.

[34] Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Society, 2022.

[35] B Favoretto, CA De Hillerin, O Bettinotti, V Oancea, and Andrea Barbarulo. Reduced order modeling via pgd for highly transient thermal evolutions in additive manufacturing. *Computer Methods in Applied Mechanics and Engineering*, 349:405–430, 2019.

[36] Kenneth R Foster, Robert Koprowski, and Joseph D Skufca. Machine learning, medical diagnosis, and biomedical engineering research-commentary. *Biomedical engineering online*, 13(1):1–9, 2014.

[37] Emily Fox and David Dunson. Multiresolution gaussian processes. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/819f46e52c25763a55cc642422644317-Paper.pdf.

[38] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 1(1):1–22, 2018.

[39] Stefania Fresca, Luca Dede', and Andrea Manzoni. A comprehensive deep learning-

based approach to reduced order modeling of nonlinear time-dependent parametrized pdes. *Journal of Scientific Computing*, 87:1–36, 2021.

[40] Aniruddha Gaikwad, Reza Yavari, Mohammad Montazeri, Kevin Cole, Linkan Bian, and Prahalada Rao. Toward the digital twin of additive manufacturing: Integrating thermal simulations, sensing, and analytics to detect process faults. *IISE Transactions*, 52(11):1204–1217, 2020.

[41] Chady Ghnatios, Khalil El Rai, Nicolas Hascoet, Pierre-Adrien Pires, Jean-Louis Duval, Jon Lambarri, Jean-Yves Hascoet, and Francisco Chinesta. Reduced order modeling of selective laser melting: from calibration to parametric part distortion. *International Journal of Material Forming*, 14:973–986, 2021.

[42] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[43] Raghav Gnanasambandam, Bo Shen, Jihoon Chung, Xubo Yue, et al. Self-scalable tanh (stan): Faster convergence and better generalization in physics-informed neural networks. *arXiv preprint arXiv:2204.12589*, 2022.

[44] Raghav Gnanasambandam, Bo Shen, Andrew Chung Chee Law, Chaoran Dou, and Zhenyu Kong. Deep gaussian process for enhanced bayesian optimization and its application in additive manufacturing. *IISE Transactions*, volume(just-accepted):1–21, 2024.

[45] John Goldak, Aditya Chakravarti, and Malcolm Bibby. A new finite element model for welding heat sources. *Metallurgical transactions B*, 15:299–305, 1984.

[46] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[47] Michael Gouge and Pan Michaleris. *Thermo-mechanical modeling of additive manufacturing*. Butterworth-Heinemann, 2017.

[48] Robert B Gramacy and Daniel W Apley. Local gaussian process approximation for large computer experiments. *Journal of Computational and Graphical Statistics*, 24 (2):561–578, 2015.

[49] Robert B Gramacy and Herbert K H Lee. Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–1130, 2008.

[50] Robert B Gramacy, Annie Sauer, and Nathan Wycoff. Triangulation candidates for bayesian optimization. *arXiv preprint arXiv:2112.07457*, 1:1–10, 2021.

[51] Stewart Greenhill, Santu Rana, Sunil Gupta, Pratibha Vellanki, and Svetha Venkatesh. Bayesian optimization for adaptive experimental design: A review. *IEEE access*, 8: 13937–13948, 2020.

[52] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. In *International conference on machine learning*, pages 2672–2680. PMLR, 2019.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[54] Ali Hebbal, Loic Brevault, Matheiu Balesdant, El-Ghazali Talbi, and Nouredine Melab. Bayesian optimization using deep gaussian processes with applications to aerospace system design, 2020.

[55] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. Nvidia simnet™: An ai-accelerated multi-physics simulation framework. In *International Conference on Computational Science*, pages 447–461. Springer, 2021.

[56] Dave Higdon, Jenise Swall, and John Kern. Non-stationary spatial modeling. *Bayesian statistics*, 6(1):761–768, 1999.

[57] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[58] WY Pauchy Hwang and Ta-You Wu. *Relativistic Quantum Mechanics and Quantum Fields: for the 21st Century*. World Scientific, 2018.

[59] Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

[60] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A*, 476(2239):20200334, 2020.

[61] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[62] Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to

forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[63] Ameya D Jagtap, Yeonjong Shin, Kenji Kawaguchi, and George Em Karniadakis. Deep kronecker neural networks: A general framework for neural networks with adaptive activation functions. *Neurocomputing*, 468:165–180, 2022.

[64] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[65] Nursultan Jyeniskhan, Aigerim Keutayeva, Gani Kazbek, Md Hazrat Ali, and Essam Shehab. Integrating machine learning model and digital twin system for additive manufacturing. *IEEE Access*, 2023.

[66] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6): 422–440, 2021.

[67] Wayne E King, Andrew T Anderson, Robert M Ferencz, Neil E Hodge, Chandrika Kamath, Saad A Khairallah, and Alexander M Rubenchik. Laser powder bed fusion additive manufacturing of metals; physics, computational, and materials challenges. *Applied Physics Reviews*, 2(4), 2015.

[68] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 22:1–43, 2014.

[69] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps

between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[70] J Nathan Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814: 1–4, 2017.

[71] Ksenia N Kyzyurova, James O Berger, and Robert L Wolpert. Coupling computer models through linking their statistical emulators. *SIAM/ASA Journal on Uncertainty Quantification*, 6(3):1151–1171, 2018.

[72] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[73] Neil Lawrence and Aapo Hyvärinen. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of machine learning research*, 6(11):611–622, 2005.

[74] Neil D Lawrence and Andrew J Moore. Hierarchical gaussian process latent variable models. In *Proceedings of the 24th international conference on Machine learning*, pages 481–488, 2007.

[75] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[76] Jungeon Lee, Hyung Jun Park, Seunghak Chai, Gyu Ri Kim, Hwanwoong Yong, Suk Joo Bae, and Daeil Kwon. Review on quality control methods in metal additive manufacturing. *Applied Sciences*, 11(4):1966, 2021.

[77] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convo-

lutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.

[78] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[79] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[80] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/JMS Journal of Data Science*, 2021.

[81] Shuheng Liao. *Toward a Digital Twin of Metal Additive Manufacturing: Process Optimization and Control Enabled by Physics-Based and Data-Driven Models*. PhD thesis, Northwestern University, 2023.

[82] Shuheng Liao, Tianju Xue, Jihoon Jeong, Samantha Webster, Kornel Ehmann, and Jian Cao. Hybrid thermal modeling of additive manufacturing processes using physics-informed neural networks for temperature prediction and parameter identification. *Computational Mechanics*, 72(3):499–512, 2023.

[83] Chao Liu, Léopold Le Roux, Carolin Körner, Olivier Tabaste, Franck Lacan, and Samuel Bigot. Digital twin-enabled collaborative data management for metal additive manufacturing systems. *Journal of Manufacturing Systems*, 62:857–874, 2022.

[84] Haitao Liu, Yew-Soon Ong, and Jianfei Cai. A survey of adaptive sampling for global

metamodeling in support of simulation-based complex engineering design. *Structural and Multidisciplinary Optimization*, 57(1):393–416, 2018.

[85] Daniel J Lizotte, Tao Wang, Michael H Bowling, Dale Schuurmans, et al. Automatic gait optimization with gaussian process regression. In *IJCAI*, volume 7, pages 944–949, 2007.

[86] Graham Loomes and Robert Sugden. Regret theory: An alternative theory of rational choice under uncertainty. *The economic journal*, 92(368):805–824, 1982.

[87] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[88] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[89] Roman Marchant and Fabio Ramos. Bayesian optimisation for intelligent environmental monitoring. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 2242–2249. IEEE, 2012.

[90] MATLAB. *R2022a)*. The MathWorks Inc., Natick, Massachusetts, 2022.

[91] Levi D McClenny and Ulisses M Braga-Neto. Self-adaptive physics-informed neural networks. *Journal of Computational Physics*, 474:111722, 2023.

[92] Lingbin Meng, Brandon McWilliams, William Jarosinski, Hye-Yeong Park, Yeon-Gil Jung, Jehyun Lee, and Jing Zhang. Machine learning in additive manufacturing: a review. *Jom*, 72:2363–2377, 2020.

[93] James L Meriam, L Glenn Kraige, and Jeff N Bolton. *Engineering mechanics: dynamics*. John Wiley & Sons, 2020.

[94] Deyu Ming and Serge Guillas. Linked gaussian process emulation for systems of computer models using mat\'ern kernels and adaptive design. *arXiv preprint arXiv:1912.09468*, 1(1):1–23, 2019.

[95] Deyu Ming, Daniel Williamson, and Serge Guillas. Deep gaussian process emulation using stochastic imputation. *Technometrics*, 65(2):150–161, 2023.

[96] Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics informed neural networks (pinns) for approximating a class of inverse problems for pdes. *arXiv preprint arXiv:2007.01138*, 2020.

[97] Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.

[98] Arvind T Mohan and Datta V Gaitonde. A deep learning based approach to reduced order modeling for turbulent flow control using lstm neural networks. *arXiv preprint arXiv:1804.09269*, 2018.

[99] Douglas C Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[100] Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Finite basis physics-informed neural networks (fbpinns): a scalable domain decomposition approach for solving differential equations. *arXiv preprint arXiv:2107.07871*, 2021.

[101] Haochen Mu, Fengyang He, Lei Yuan, Philip Commins, Hongmin Wang, and Zengxi Pan. Toward a smart wire arc additive manufacturing system: A review on current developments and a framework of digital twin. *Journal of Manufacturing Systems*, 67: 174–189, 2023.

[102] Iain Murray, Ryan Adams, and David MacKay. Elliptical slice sampling. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 541–548. JMLR Workshop and Conference Proceedings, 2010.

[103] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[104] Paromita Nath and Sankaran Mahadevan. Probabilistic digital twin for additive manufacturing process design and control. *Journal of Mechanical Design*, 144(9):091704, 2022.

[105] Christopher J Paciorek and Mark J Schervish. Spatial modelling using a new class of nonstationary covariance functions. *Environmetrics: The official journal of the International Environmetrics Society*, 17(5):483–506, 2006.

[106] Minas Pantelidakis, Konstantinos Mykoniatis, Jia Liu, and Gregory Harris. A digital twin ecosystem for additive manufacturing using a real-time development platform. *The International Journal of Advanced Manufacturing Technology*, 120(9-10):6547–6563, 2022.

[107] Athanasios Papoulis and S Unnikrishna Pillai. *Probability, random variables and stochastic processes*. McGraw-Hill: Boston., 2002.

[108] Suraj Pawar, Sk Mashfiqur Rahman, H Vaddireddy, Omer San, Adil Rasheed, and Prakash Vedula. A deep learning enabler for nonintrusive reduced order modeling of fluid flows. *Physics of Fluids*, 31(8), 2019.

[109] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,

M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[110] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

[111] Rahul Rai, Manoj Kumar Tiwari, Dmitry Ivanov, and Alexandre Dolgui. Machine learning in manufacturing and industry 4.0 applications, 2021.

[112] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378: 686–707, 2019.

[113] Dushhyanth Rajaram, Tejas G Puranik, Ashwin Renganathan, Woong Je Sung, Olivia J Pinon-Fischer, Dimitri N Mavris, and Arun Ramamurthy. Deep gaussian process enabled surrogate models for aerodynamic flows. In *AIAA Scitech 2020 Forum*, page 1640, 2020.

[114] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 13:1–21, 2017.

[115] Chitta Ranjan. *Understanding deep learning: Application in rare event prediction*. Connaissance Publishing, 2020.

[116] Adil Rasheed, Omer San, and Trond Kvamsdal. Digital twin: Values, challenges and enablers from a modeling perspective. *Ieee Access*, 8:21980–22012, 2020.

[117] Carl Edward Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. *Advances in neural information processing systems*, 2:881–888, 2002.

[118] Carl Edward Rasmussen and Christopher K Williams. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[119] David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *ACM Computing Surveys (CSUR)*, 55(2):1–96, 2022.

[120] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[121] Elizabeth G Ryan, Christopher C Drovandi, James M McGree, and Anthony N Pettitt. A review of modern computational algorithms for bayesian optimal design. *International Statistical Review*, 84(1):128–154, 2016.

[122] Francisco Sahli Costabal, Yibo Yang, Paris Perdikaris, Daniel E Hurtado, and Ellen Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8:42, 2020.

[123] Amirul Islam Saimon, Emmanuel Yangue, Xiaowei Yue, Chenang Liu, et al. Advancing additive manufacturing through deep learning: A comprehensive review of current progress and future challenges. *arXiv preprint arXiv:2403.00669*, 2024.

[124] Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep gaussian processes. *arXiv preprint arXiv:1705.08933*, 1(1):1–16, 2017.

[125] Esteban Samaniego, Cosmin Anitescu, Somdatta Goswami, Vien Minh Nguyen-Thanh, Hongwei Guo, Khader Hamdia, X Zhuang, and T Rabczuk. An energy approach to

the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering*, 362:112790, 2020.

[126] Gabriel Avelino Sampedro, Made Adi Paramartha Putra, and Mideth Abisado. 3d-amplifai: An ensemble machine learning approach to digital twin fault monitoring for additive manufacturing in smart factories. *IEEE Access*, 2023.

[127] Annie Sauer, Andrew Cooper, and Robert B Gramacy. Vecchia-approximated deep gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics*, 1:1–14, 2022.

[128] Annie Sauer, Robert B Gramacy, and David Higdon. Active learning for deep gaussian process surrogates. *Technometrics*, 1(1):1–15, 2022.

[129] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[130] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczynski. *Lectures on stochastic programming: modeling and theory*. SIAM, 2021.

[131] Maulshree Singh, Evert Fuenmayor, Eoin P Hinchy, Yuansong Qiao, Niall Murray, and Declan Devine. Digital twin: Origin to future. *Applied System Innovation*, 4(2):36, 2021.

[132] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 1: 1–12, 2012.

[133] Jasper Snoek, Kevin Swersky, Rich Zemel, and Ryan Adams. Input warping for bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*, pages 1674–1682. PMLR, 2014.

[134] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 1(1):1–17, 2009.

[135] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved December 17, 2021, from http://www.sfu.ca/~ssurjano, 2021.

[136] Akinori Tanaka, Akio Tomiya, and Kōji Hashimoto. *Deep Learning and Physics*. Springer, 2021.

[137] Fei Tao, Bin Xiao, Qinglin Qi, Jiangfeng Cheng, and Ping Ji. Digital twin modeling. *Journal of Manufacturing Systems*, 64:372–389, 2022.

[138] Søren Taverniers, Svyatoslav Korneev, Kyle M Pietrzyk, and Morad Behandish. Accelerating part-scale simulation in liquid metal jet additive manufacturing via operator learning. *arXiv preprint arXiv:2202.03665*, 2022.

[139] Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

[140] Michalis Titsias and Neil D Lawrence. Bayesian gaussian process latent variable model. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 844–851. JMLR Workshop and Conference Proceedings, 2010.

[141] David John James Toal and Andy J Keane. Non-stationary kriging for design optimization. *Engineering Optimization*, 44(6):741–765, 2012.

[142] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[143] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[144] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384: 113938, 2021.

[145] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[146] Sarah Webb. Deep learning for biology. *Nature*, 554(7690):555–558, 2018.

[147] Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Fre-

quency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523*, 2019.

[148] Mahmoud Yaseen, Dewen Yushu, Peter German, and Xu Wu. Fast and accurate reduced-order modeling of a moose-based additive manufacturing model with operator learning. *The International Journal of Advanced Manufacturing Technology*, 129(7): 3123–3139, 2023.

[149] Dani Yogatama and Noah A Smith. Bayesian optimization of text representations. *arXiv preprint arXiv:1503.00693*, 1:1–9, 2015.

[150] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, and Dionisis Kandris. A review of machine learning and iot in smart transportation. *Future Internet*, 11(4):94, 2019.

[151] Wengang Zhang, Hongrui Li, Yongqin Li, Hanlong Liu, Yumin Chen, and Xuanming Ding. Application of deep learning algorithms in geotechnical engineering: a short critical review. *Artificial Intelligence Review*, 54(8):5633–5673, 2021.

[152] Qiming Zhu, Zeliang Liu, and Jinhui Yan. Machine learning for metal additive manufacturing: Predicting temperature and melt pool fluid dynamics using physics-informed neural networks. *Computational Mechanics*, 67(2):619–635, 2021.

[153] Tarek I Zohdi. *Modeling and Simulation of Advanced Manufacturing Processes*. Springer, 2014.

[154] Ran Zou, Xuan Liang, Qian Chen, Mohan Wang, Mohamed AS Zaghloul, Hui Lan, Michael P Buric, Paul R Ohodnicki, Benjamin Chorpening, Albert C To, et al. A digital twin approach to study additive manufacturing processing using embedded

optical fiber sensors and numerical modeling. *Journal of Lightwave Technology*, 38 (22):6402–6411, 2020.

[155] Daniel Zwillinger and Vladimir Dobrushkin. *Handbook of differential equations*. Chapman and Hall/CRC, 1998.

Appendices

Appendix A

Appendix for Chapter 3

A.1 DGP-SI Training Example

This section provides a brief explanation of the DGP training using Stochastic Imputation based on a 3-layer DGP as shown in Figure A.1.

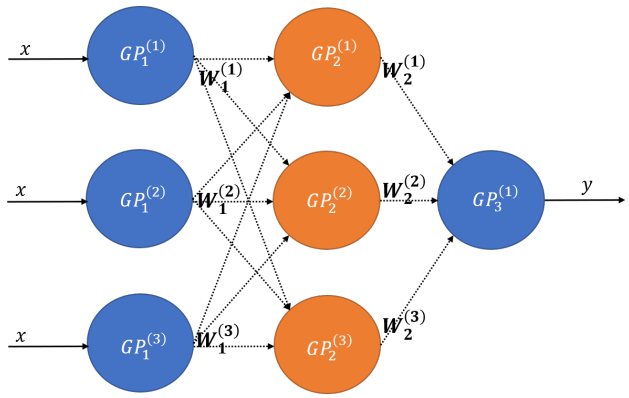


Figure A.1: Inferring a 3-layered DGP with SI involves imputing the latent variables $\mathbf{W}$ multiple times using MCMC. For a realization of $\mathbf{W}$, this structure (with known $\mathbf{W}$) is called the LGP. Blue circles indicate at least one of the input and output is observed whereas orange circles indicate both the input and output are inferred.

For the DGP in Figure A.1, $W_l^{(\alpha)}\ \forall\ \alpha \in \{1,2,3\}$ are assumed to be conditionally independent, and all $W_l^{(\alpha)} \in \mathbb{R}^{n\times 1}\ \forall\ l \in \{1,2\}, \alpha \in \{1,2,3\}$. To infer this, an ESS within a Gibbs sampler is used with the prior on $W_l^{(\alpha)}\ \forall\ l \in \{1,2\}, \alpha \in \{1,2,3\}$ as a multivariate Gaussian [95]. Thus, we get the realizations of $p(W_l^{(\alpha)}|\mathbf{X},\mathbf{y})\ \forall\ l \in \{1,2\}, \alpha \in \{1,2,3\}$. Using these realizations, the corresponding $\Theta_m^{(\gamma)}\ \forall\ m \in \{1,2\}, \gamma \in \{1,2,3\}$ and $\Theta_3^{(1)}$ can be cal-

culated by maximizing the likelihood. The imputation step and the maximization step are repeated typically for a fixed number of iterations to get the optimal $\boldsymbol{\Theta}^*$.

A.2 Illustration of bagging

The illustration of bagging in the optimization of 1D function (defined in Section 2.3.1) is provided in Figure A.2. For each realization $\mathbf{W}_{(t)}$ of $\mathbf{W}$, there is a realization $a_{y_{(t)}}(x)$ of $a_y(x)$. Among the 50 realizations (in this illustrative example), R (which is again taken as 50) number of realizations are chosen, with replacement after every selection, to form one bagging set. The bagging is repeated for B times (taken as 5) to get 5 sets of 50 "pseudo"-realizations. The bagging procedure brings about 5 optimal values $\hat{x}_b \; \forall \; b = 1, 2, 3, 4, 5$. These $\hat{x}_b \; \forall \; b = 1, 2, 3, 4, 5$ are averaged to obtain the next evaluation point at every step. These $a_{y_{(t)}}(x) \; \forall \; t = \{1, 2, 3, ..., 50\}$ are shown as realizations of $a_y(x)$ in Figure A.2. Though the value of B will be typically higher than 5 (few hundreds were used in the case studies provided in the main text), only five sets are shown in Figure A.2 for the purpose of illustration.

A.3 Analytical Case Studies

A.3.1 Function Equations

Several analytical functions are used to test the case studies provided in main text. These functions are typically used for testing optimization algorithms in literature [135]. The functions used in this work with their equations are as below. ***Six-Hump Camel:*** $f(x) = (4 - 2.1\mathbf{x}_1^2 + \frac{\mathbf{x}_1^4}{3})\mathbf{x}_1^2 + \mathbf{x}_1\mathbf{x}_2 + (-4 + 4\mathbf{x}_2^2)\mathbf{x}_2^2$;

Three-Hump Camel: $f(x) = 2\mathbf{x}_1^2 - 1.05\mathbf{x}_1^4 + \frac{\mathbf{x}_1^6}{6} + \mathbf{x}_1\mathbf{x}_2 + \mathbf{x}_2^2$;

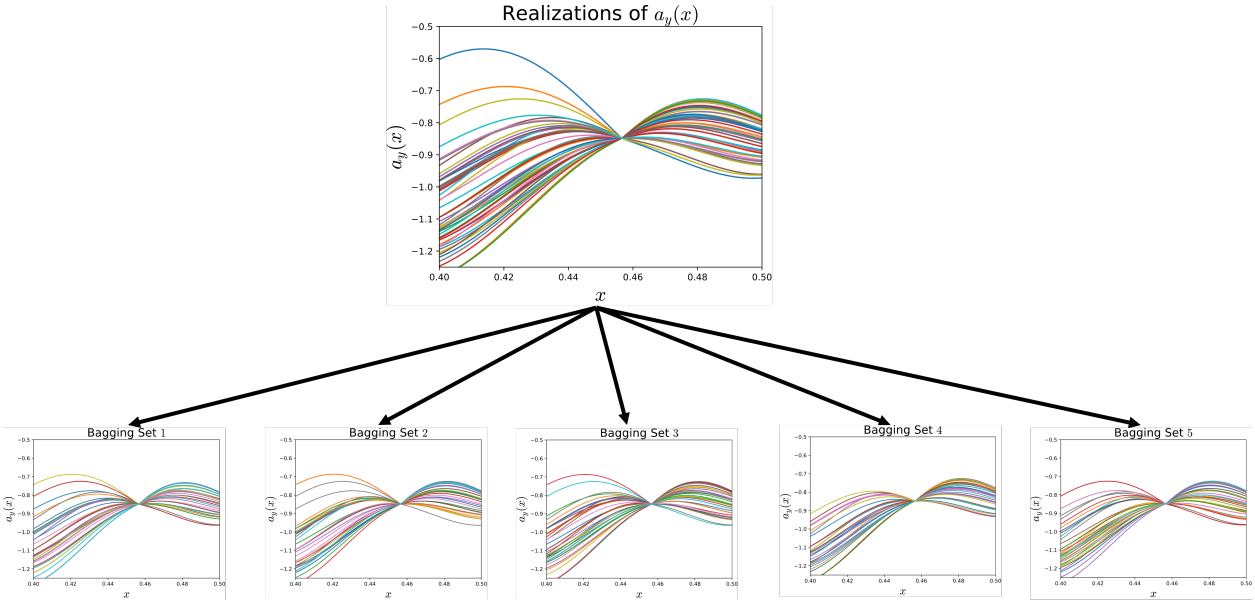


Figure A.2: Illustration of bagging with $B = 5$ and with $R = T = 50$ for the 1D test function on a small subset of the support. The 25 realizations are sampled with replacement 50 times to obtain the each bagging set shown.

Hartmann 3D: $f(x) = -\sum_{i=1}^{4} \alpha_i \exp(-\sum_{j=1}^{3} A_{ij}(\mathbf{x}_j - P_{ij})^2)$;

Hartmann 4D: $f(x) = \frac{1}{0.839}[1.1 - \sum_{i=1}^{4} \alpha_i \exp(-\sum_{j=1}^{4} A_{ij}(\mathbf{x}_j - P_{ij})^2)]$,

$\mathbf{x}_i$ - input x in i^{th} dimension. α, A, P - constants [135].

Michalewicz 10D: $f(x) = -\sum_{i=1}^{10} \sin(x_i) \sin^{2m}(\frac{ix_i^2}{\pi})$

$\mathbf{x}_i$ - input x in i^{th} dimension. m - constant $(= 10)$ [135].

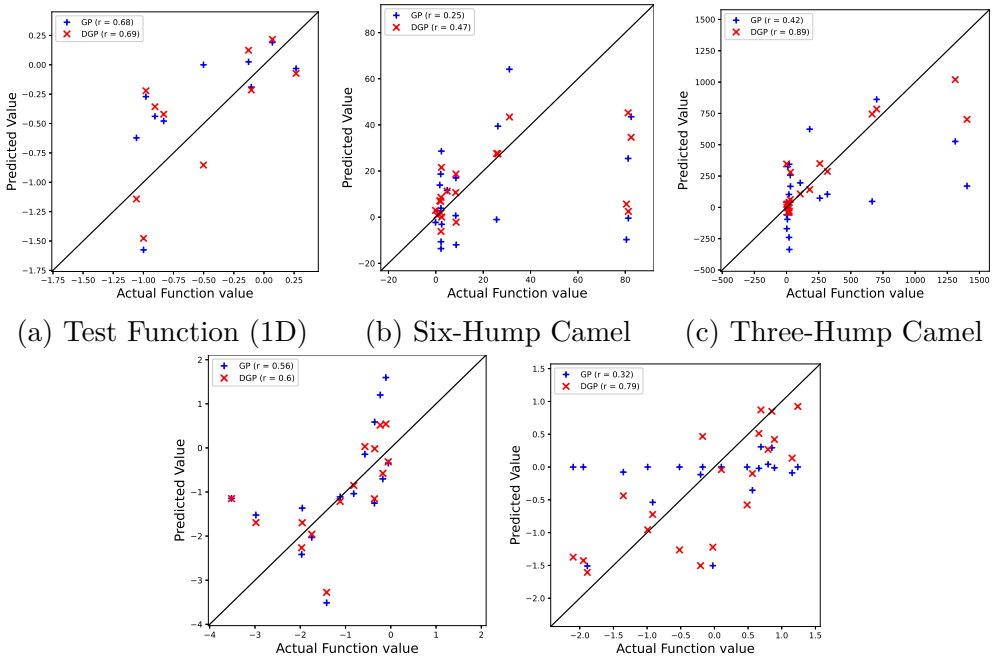
A.3.2 Comparing GP and DGP modeling

The analytical functions are tested for prediction capability in comparison with the GP, to show the advantage of modeling with the DGP. A set of $5 \times$ *dimension* or $10 \times$ *dimension* points are chosen through Latin Hypercube Sampling (LHS) from the support. The points are checked for a correlation between the predicted and actual value by Leave One Out Cross Validation (LOOCV). It is observed that the predictions of the DGP are, in general,

less spread around the 45° line showing better predictions than the GP. Table A.1 shows the average correlation values for the five random sets of points and Figure A.3 shows the correlation plot.

Table A.1: Average LOOCV correlation values of five random set of point comparing the GP model and the DGP model.

Function Name	**DGP-SI** [95]	**GP** [118]
Test Function (1D)	0.3634	0.1002
6-Hump Camel	0.1734	0.1545
3-Hump Camel	0.2578	0.1622
Hartmann3	0.6362	0.4391
Hartmann4	0.3303	0.0842



(a) Test Function (1D) (b) Six-Hump Camel (c) Three-Hump Camel

(d) Hartmann3 (e) Hartmann4

Figure A.3: Correlation plot for comparing the prediction capabilities of a GP model and a DGP model. The LOOCV predictions for all the chosen points for each of the functions are shown. The 45° line shows an ideal model prediction where predicted values and the actual function values are the same.

A.3.3 BO Implementation Details

There are several required choices for each of the implemented algorithms (including the proposed algorithm). The parameters that are utilized for the case studies are provided in Table A.2. For GP-BO and WGP-BO, n_{opt} denotes the number of optimization restarts for the hyperparameters. lr denotes the learning rate of Adam optimizer for DSVI of DGP. n_{epoch}, n_{mcmc}, and N denote the training iterations (or samples for DGP-MCMC-BO) of the corresponding algorithms. The subscript or superscript i or $init$ denotes the initial number of training iterations (or samples). The initial training iterations (or samples) are usually longer following [128]. All the DGP surrogates utilize a 2-layer GP with the number of GPs in the first-layer equaling the dimensionality of input, and 1 GP in the second layer. For the proposed algorithm, T, R, and B denote the number of prediction samples, the size of bagging, and the number of bagging sets, respectively. IC denotes the *Input-Connection*, which is a feature that is used additionally on the SI algorithm to connect the inputs to any intermediate layer (in the case studies, it is the last layer). See [95] for more details on IC.

A.3.4 Comparisons between EI and UCB

Figure A.4 shows the direct comparisons made between UCB and EI on some of the analytical test functions. Though UCB is empirically better unless we use a fixed candidate set (not shown). For the cases utilizing fixed candidate set (*Michalewicz 10D* and AM simulation case study), EI was used due to better performance.

Table A.2: The technical choices of all the algorithms that are used for showcasing the proposed DGP-SI-BO algorithm.

	General Settings	GP-BO [38]	WGP-BO [133]	DGP-DSVI-BO [54]	DGP-MCMC-BO [50]	**DGP-SI-BO (Proposed)**
Test Function (1D)	$n_{steps} = 100$ $n_{iters} = 10$	$n_{opt} = 10$	$n_{opt} = 10$	$lr = 0.01$ $n^i_{epoch} = 500, n_{epoch} = 150$ 2-layers - 1 GP each	$n^i_{mcmc} = 10000$ (80% burn-in) $n_{mcmc} = 3000$ (33% burn-in) 2-layers - 1 GP each	$N_{init} = 500$ (75% burn-in) $N = 150$ (50% burn-in) $T = 50, R = 50, B = 100$ 2-layers - 1 GP each IC = $True$
Six-hump Camel	$n_{steps} = 150$ $n_{iters} = 10$	$n_{opt} = 10$	$n_{opt} = 10$	$lr = 0.01$ $n^i_{epoch} = 500, n_{epoch} = 150$ 2-layers - {2,1} GPs	$n^i_{mcmc} = 10000$ (80% burn-in) $n_{mcmc} = 3000$ (33% burn-in) 2-layers - {2,1} GPs	$N_{init} = 500$ (75% burn-in) $N = 150$ (50% burn-in) $T = 50, R = 50, B = 400$ 2-layers - {2,1} GPs IC = $True$
Three-hump Camel	$n_{steps} = 150$ $n_{iters} = 10$	$n_{opt} = 10$	$n_{opt} = 10$	$lr = 0.01$ $n^i_{epoch} = 500, n_{epoch} = 150$ 2-layers - {2,1} GPs	$n^i_{mcmc} = 10000$ (80% burn-in) $n_{mcmc} = 3000$ (33% burn-in) 2-layers - {2,1} GPs	$N_{init} = 500$ (75% burn-in) $N = 150$ (50% burn-in) $T = 50, R = 50, B = 400$ 2-layers - {2,1} GPs IC = $True$
Hartmann3	$n_{steps} = 200$ $n_{iters} = 10$	$n_{opt} = 10$	$n_{opt} = 10$	$lr = 0.01$ $n^i_{epoch} = 500, n_{epoch} = 150$ 2-layers - {3,1} GPs	$n^i_{mcmc} = 10000$ (80% burn-in) $n_{mcmc} = 3000$ (33% burn-in) 2-layers - {3,1} GPs	$N_{init} = 500$ (75% burn-in) $N = 150$ (50% burn-in) $T = 50, R = 50, B = 400$ 2-layers - {3,1} GPs IC = $True$
Hartmann4	$n_{steps} = 200$ $n_{iters} = 10$	$n_{opt} = 10$	$n_{opt} = 10$	$lr = 0.01$ $n^i_{epoch} = 500, n_{epoch} = 150$ 2-layers - {4,1} GPs	$n^i_{mcmc} = 10000$ (80% burn-in) $n_{mcmc} = 3000$ (33% burn-in) 2-layers - {4,1} GPs	$N_{init} = 500$ (75% burn-in) $N = 150$ (50% burn-in) $T = 50, R = 50, B = 400$ 2-layers - {4,1} GPs IC = $True$
Michalewicz	$n_{steps} = 200$ $n_{iters} = 5$	$n_{opt} = 10$	$n_{opt} = 10$	$lr = 0.01$ $n^i_{epoch} = 500, n_{epoch} = 150$ 2-layers - {10,1} GPs	$n^i_{mcmc} = 5000$ (80% burn-in) $n_{mcmc} = 1500$ (33% burn-in) 2-layers - {10,1} GPs	$N_{init} = 400$ (75% burn-in) $N = 150$ (50% burn-in) $T = 50, R = 50, B = 400$ 2-layers - {10,1} GPs IC = $True$

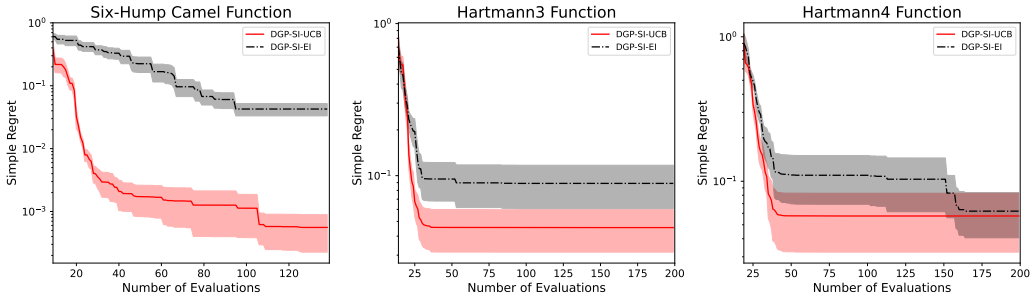


Figure A.4: Comparison of the UCB and EI acquisition functions used in the same settings of the DGP-SI inference on few analytical test functions. For all the cases shown here, UCB is better but EI is particularly significant when using a fixed candidate dataset.

A.4 AM Simulation Case Study

A.4.1 Correlation Study

A LOOCV procedure was conducted on the initial 15 LHS samples of the AM simulation case study. Figure A.5 shows the correlation plot with the correlation values. The DGP-SI is poor in predictions with the LHS samples, but eventually gets better to optimize the function better.

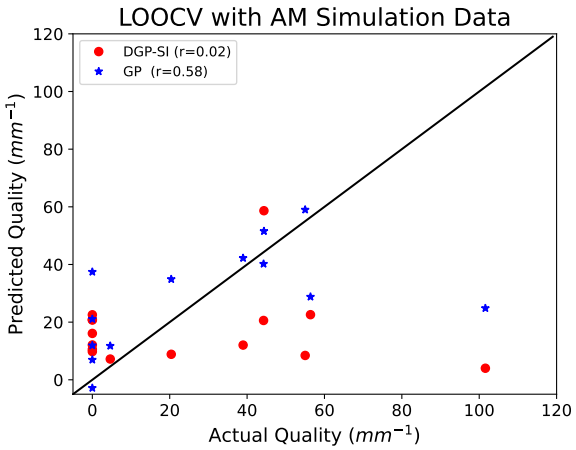


Figure A.5: Comparison of the UCB and EI acquisition functions used in the same settings of the DGP-SI inference on few analytical test functions. For all the cases shown here, UCB is better but EI is particularly significant when using a fixed candidate dataset.

A.4.2 BO Implementation Details

There are several required technical choices for each of the implemented algorithms (including the proposed algorithm). The parameters that are utilized for the AM simulation case studies are provided in Table A.3. The parameter definitions are provided in Appendix A.3.3.

Table A.3: The technical choices of all the algorithms used in this work for the AM simulation case study. Note that DGP-MCMC-BO uses fewer MCMC samples than the numerical case studies but the algorithm is still slower than the proposed method (as provided in Table 2.3

General Settings	GP-BO [38]	WGP-BO [133]	DGP-DSVI-BO [54]	DGP-MCMC-BO [50]	**DGP-SI-BO (Proposed)**
$n_{steps} = 150$ $n_{iters} = 5$	$n_{opt} = 10$	$n_{opt} = 10$	$lr = 0.001$ $n^i_{epoch} = 500$,$n_{epoch} = 150$ 3-layers - {3,3,1} GPs	$n^i_{mcmc} = 3000$ (83.33% burn-in) $n_{mcmc} = 1000$ (33% burn-in) 3-layers - {3,3,1} GPs	$N_{init} = 500$ (75% burn-in) $N = 150$ (50% burn-in) $T = 25$,$R = 25$,$B = 400$ 3-layers - {3,3,1} GPs IC = $False$

Appendix B

Appendix for Chapter 4

B.1 Derivation for derivatives of NN

Derivation for the first derivative for NN follows the notations of Eq.(2) in main text. The derivation is a straightforward application of chain rule of differentiation. Considering the last (linear) layer alone,

$$u^{(1)}_{\mathbf{\Theta}}(x_i) = \sum_{p=1}^{P} \mathbf{W}^p_{D+1} \frac{\partial(\mathbf{z}^p_D)}{\partial x}|_{x=x_i}, \tag{B.1}$$

where the p in $\mathbf{W}^p_{D+1}$ and $\mathbf{z}^p_D$ denotes the particular element in $\mathbf{W}_{D+1}$ and $\mathbf{z}_D$ respectively, indexed by p.

$$\frac{\partial(\mathbf{z}^p_D)}{\partial x}|_{x=x_i} = \sigma^{'}(\mathbf{L}^p_D) \frac{\partial \mathbf{L}^p_D}{\partial x}|_{x=x_i} \tag{B.2}$$

Further,

$$\begin{aligned} \frac{\partial \mathbf{L}^p_D}{\partial x}|_{x=x_i} &= \frac{\partial(\sum_{q=1}^{P}[\mathbf{W}^{qp}_D \mathbf{z}^q_{D-1} + \mathbf{b}^{qp}])}{\partial x}|_{x=x_i} \\ \frac{\partial \mathbf{L}^p_D}{\partial x}|_{x=x_i} &= \sum_{q=1}^{P} \mathbf{W}^{qp}_D \frac{\partial(\mathbf{z}^q_{D-1})}{\partial x}|_{x=x_i}, \end{aligned} \tag{B.3}$$

where q denote the index in $(D-1)^{th}$ layer. Eq. (B.3) is similar to Eq. (B.1) and combining Eqs. (1)-(3),

$$u^{(1)}_{\mathbf{\Theta}}(x_i) = \sum_{p=1}^{P} \Big[\mathbf{W}^p_{D+1} \sigma^{'}(\mathbf{L}^p_D) \Big(\sum_{q=1}^{P} \mathbf{W}^{qp}_D \frac{\partial(\mathbf{z}^q_{D-1})}{\partial x}|_{x=x_i} \Big) \Big]. \tag{B.4}$$

From Eq.(B.4), it is easy to see the recurrence relationship over the layers. Introducing some notation and generalizing the equation for any x,

$$\mathbf{G}_k = \begin{bmatrix} \frac{\partial(\mathbf{z}^1_k)}{\partial x} \\ \frac{\partial(\mathbf{z}^2_k)}{\partial x} \\ \frac{\partial(\mathbf{z}^3_k)}{\partial x} \\ . \\ . \\ . \\ \frac{\partial(\mathbf{z}^P_k)}{\partial x} \end{bmatrix} \quad \forall \; k = \{1, 2, 3, ..., D\} \tag{B.5}$$

$$u^{(1)}_{\mathbf{\Theta}}(x) = \sum_{p=1}^{P} \Big[\mathbf{W}^p_{D+1} \sigma^{'}(\mathbf{L}^p_D) \Big(\mathbf{W}^{(p)}_D \mathbf{G}_{D-1} \Big) \Big],$$

where $\mathbf{W}^{(p)}_D (\in \mathbb{R}^{1 \times P})$ denote the all the weights connecting to node index p from the previous layer. Let,

$$\mathbf{H}_k = \mathbf{W}_k \mathbf{G}_{k-1} \; \forall \; k = \{2, 3, ..., D\}.$$

Then,

$$u^{(1)}_{\mathbf{\Theta}}(x) = \sum_{p=1}^{P} \mathbf{W}^p_{D+1} \sigma^{'}(\mathbf{L}^p_D) \mathbf{H}^p_k.$$

Note that,

$$\frac{\partial(\mathbf{z}_D^p)}{\partial x} = \sigma^{'}(\mathbf{L}_D^p)\mathbf{H}_k^p$$
$$\implies \mathbf{G}_D = \sigma^{'}(\mathbf{L}_D) \circledcirc \mathbf{H_D}$$

where $\circledcirc$ denotes the element-wise multiplication operator.

Then,

$$u_{\mathbf{\Theta}}^{(1)}(x) = \sum_{p=1}^{P} \mathbf{W}_{D+1}^p \mathbf{G}_D^p$$
$$\implies u_{\mathbf{\Theta}}^{(1)}(x) = \mathbf{W}_{D+1}\mathbf{G}_D.$$

Further,

$$\mathbf{G}_1^p = \frac{\partial(\mathbf{z}_1^p)}{\partial x}$$
$$\implies \mathbf{G}_1^p = \frac{\partial(\sigma(\mathbf{W}_1^p x + b_1^p))}{\partial x}$$
$$\implies \mathbf{G}_1^p = \sigma'(\mathbf{L}_1^p)\frac{\partial(\mathbf{W}_1^p x + b_1^p)}{\partial x}$$
$$\implies \mathbf{G}_1^p = \sigma'(\mathbf{L}_1^p)\mathbf{W}_1^p$$
$$\implies \mathbf{G}_1 = \sigma'(\mathbf{L}_1) \circledcirc \mathbf{W}_1$$

Thus, defining $\mathbf{H}_1 = \mathbf{W}_1$, we get the final recurrence equation of calculating the derivative

of a simple NN. Figure B.1 provides a way to visualize the function.

$$\begin{aligned} \mathbf{H}_1 &= \mathbf{W}_1 \\ \mathbf{G}_i &= \sigma^{'}(\mathbf{L_i}) \circledcirc \mathbf{H}_i \ \forall \ i \in \{1, 2, 3, ..., D\} \\ \mathbf{H}_i &= \mathbf{W}_i \mathbf{G}_{i-1} \ \forall \ i \in \{2, 3, ..., D+1\} \\ u_{\mathbf{\Theta}}(x) &= \mathbf{H}_{D+1} \end{aligned} \tag{B.6}$$

Similarly, we get the final recurrence equation of calculating the second derivative of a simple NN.

$$\begin{aligned} \mathbf{F}_1 &= \mathbf{0} \\ \mathbf{C}_i &= \sigma^{''}(\mathbf{L_i}) \circledcirc \mathbf{H}_i \circledcirc \mathbf{H}_i \ \forall \ i \in \{1, 2, 3, ..., D\} \\ \mathbf{E}_i &= \mathbf{C}_i + \sigma^{'}(\mathbf{L_i}) \circledcirc \mathbf{F}_i \ \forall \ i \in \{1, 2, 3, ..., D\} \\ \mathbf{F}_i &= \mathbf{W}_i \mathbf{E}_{i-1} \ \forall \ i \in \{2, 3, ..., D+1\} \\ u^{(2)}_{\mathbf{\Theta}}(x) &= \mathbf{F}_{D+1} \end{aligned} \tag{B.7}$$

B.2 Expectation and Variance of tanh derivatives

As mentioned in the main text, let the output of tanh follow a normal distribution centered around zero[42, 75]. Let k_0 denote this normally distributed random variable with a standard deviation s.

$$\begin{aligned} k_0 &\sim \mathcal{N}(0, s^2) \\ \mathbb{E}[k_0] &= 0 \\ \mathbb{V}[k_0] &= s^2 \end{aligned} \tag{B.8}$$

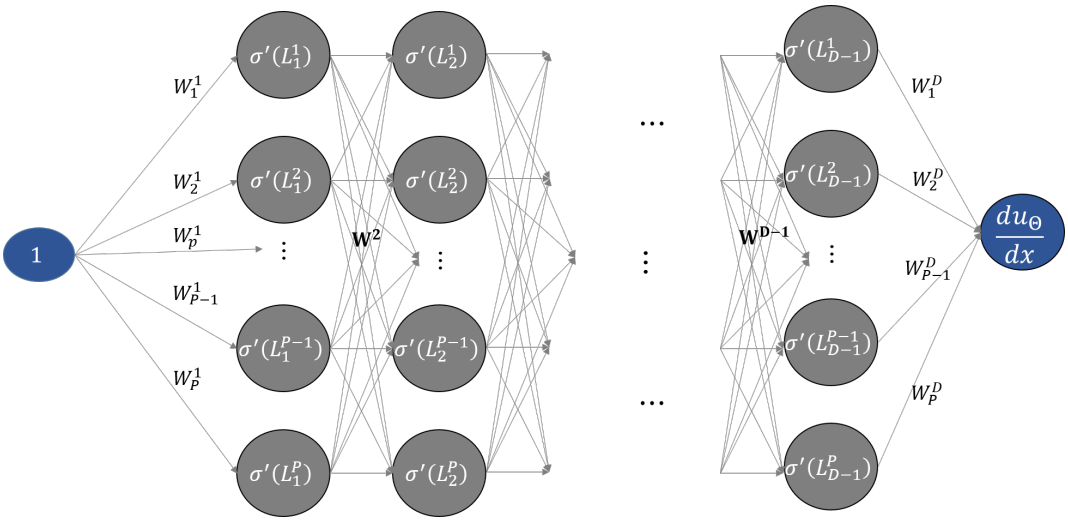


Figure B.1: Visualizing the calculation of derivative of output of a PINN with respect to its input. The nodes here are constant values of the derivative function of the activation at the specified $\mathbf{L}_k^p$. The constant valued nodes are multiplied to its input. The input to the network is the constant value 1 and the output is the value of derivative calculated from the PINN. Note that this is **not a Neural Network** but rather a representation to throw light on the derivative calculation in the loss function. The values of the "nodes" are dependent on the particular value of x.

The moments of a normally distributed random variables are well studied[107]. Particularly,

$$\mathbb{E}[k_0^n] = \begin{cases} 0, & \text{if } n \text{ is odd} \\ s^n(n-1)!!, & \text{if } n \text{ is even} \end{cases} \tag{B.9}$$

Let k_i denote the i^{th} derivative of tanh. It is well-known that the first derivative of tanh is $1 - \tanh^2$.

$$k_1 = 1 - k_0^2 \tag{B.10}$$

$$\begin{aligned} \mathbb{E}[k_1] &= \mathbb{E}[1 - k_0^2] \\ \implies \mathbb{E}[k_1] &= 1 - \mathbb{E}[k_0^2] \\ \implies \mathbb{E}[k_1] &= 1 - s^2 \end{aligned}$$

$$\begin{aligned} \mathbb{V}[k_1] &= \mathbb{E}[k_1^2] - (\mathbb{E}[k_1])^2 \\ &= \mathbb{E}[(1 - k_0^2)^2] - (1 - s^2)^2 \\ &= \mathbb{E}[1 + k_0^4 - 2k_0^2] - (1 - s^2)^2 \\ &= 1 + \mathbb{E}[k_0^4] - 2\mathbb{E}[k_0^2] - (1 + s^4 - 2s^2) \\ &= 1 + s^4(3) - 2s^2 - (1 + s^4 - 2s^2) \\ \implies \mathbb{V}[k_1] &= 2s^4 \end{aligned}$$

The second derivative is given as,

$$\begin{aligned} k_2 &= (1 - \tanh^2(x))' \\ k_2 &= -2\tanh(x)(1 - \tanh^2(x)) \\ \implies k_2 &= -2\tanh(x) + 2\tanh^3(x) \\ \implies k_2 &= -2k_0 + 2k_0^3 \end{aligned} \tag{B.11}$$

$$\begin{aligned} \mathbb{E}[k_2] &= -2\mathbb{E}[k_0] + 2\mathbb{E}[k_0^3] \\ \implies \mathbb{E}[k_2] &= 0 \end{aligned}$$

$$
\begin{aligned}
\mathbb{V}[k_2] &= \mathbb{E}[k_2^2] - (\mathbb{E}[k_2])^2 \\
&= \mathbb{E}[4(-k_0 + k_0^3)^2] - 0 \\
&= 4\mathbb{E}[k_0^2 + k_0^6 - 2k_0k_0^3] \\
&= 4(\mathbb{E}[k_0^2] + \mathbb{E}[k_0^6] - 2\mathbb{E}[k_0^4]) \\
&= 4(s^2 + s^6(5 \times 3) - 2s^4(3)) \\
\implies \mathbb{V}[k_2] &= 4s^2(15s^4 - 6s^2 + 1)
\end{aligned}
$$

From Eq. (B.11), the third derivative of tanh,

$$
\begin{aligned}
k_3 &= -2(\tanh(x))' + 2(\tanh^3(x))' \\
k_3 &= -2(1 - \tanh^2(x)) + 2(3\tanh^2(x)(1 - \tanh^2(x))) \\
k_3 &= -2 + 2\tanh^2(x) + 6\tanh^2(x) - 6\tanh^4(x) \\
\implies k_3 &= -2 + 8\tanh^2(x) - 6\tanh^4(x) \\
\implies k_3 &= -2 + 8k_0^2 - 6k_0^4
\end{aligned}
\tag{B.12}
$$

$$
\begin{aligned}
\mathbb{E}[k_3] &= -2 + 8\mathbb{E}[k_0^2] - 6\mathbb{E}[k_0^4] \\
\implies \mathbb{E}[k_3] &= -2 + 8s^2 - 6(3s^4) \\
\implies \mathbb{E}[k_3] &= -2 + 8s^2 - 18s^4
\end{aligned}
$$

B.3 Derivatives of Activation Functions during training

Figure B.2 shows the evolution of the σ, $\sigma^{'}$, and $\sigma^{''}$ in solving a second-order ODE with the activation functions used in this work. From the first-derivative plots, as expected, tanh tends to move away from $\sigma^{'}$ value of 1 as the training proceeds. The Swish activation functions' standard deviation remains low throughout the training. Rowdy activation function has low standard deviation in Layer 1 and very high standard deviation in Layer 2 (not shown in plot).

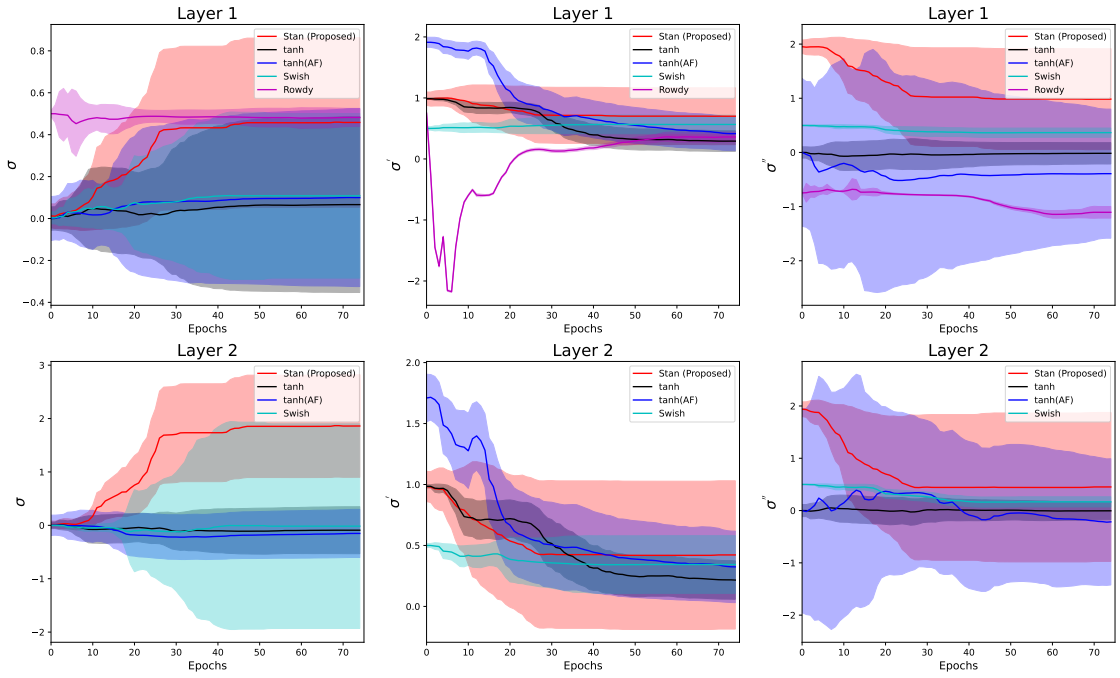


Figure B.2: The mean (with one-standard deviation error-bar) of activation function, its first and second derivative over the training epochs for 50-units width 2-layer PINN. Rowdy activation function is skipped in Layer 2 due to large standard deviation (of order 10^1) in the course of training.

B.4 Gradient of Loss Function for Multi-layer NN

Here, we derive the gradients of loss with respect to the weights in the last two layers. Consider the loss function in Eq. (6) in the main text.

$$\begin{aligned} J(\mathbf{\Theta}) =& \frac{1}{N_f}\sum_{i=1}^{N_f}\Big(\frac{d^2u_{\mathbf{\Theta}}}{dx^2}|_{x_i} + \frac{du_{\mathbf{\Theta}}}{dx}|_{x_i} - 6u_{\mathbf{\Theta}}(x_i)\Big)^2 \\ &+ \Big(u_{\mathbf{\Theta}}(0) - 2\Big)^2 \\ &+ \Big(\frac{du_{\mathbf{\Theta}}}{dx}|_{x=0} - (-1)\Big)^2 \end{aligned} \tag{B.13}$$

To establish the effect of activation function in the training, we try to derive the expression explicitly. Let,

$$\begin{aligned} J_1(\mathbf{\Theta}) &= \frac{1}{N_f}\sum_{i=1}^{N_f}\Big(\frac{d^2u_{\mathbf{\Theta}}}{dx^2}|_{x_i} + \frac{du_{\mathbf{\Theta}}}{dx}|_{x_i} - 6u_{\mathbf{\Theta}}(x_i)\Big)^2 \\ \frac{\partial J_1(\mathbf{\Theta})}{\partial\theta} =& \frac{2}{N_f}\sum_{i=1}^{N_f}\Big(\frac{d^2u_{\mathbf{\Theta}}}{dx^2}|_{x_i} + \frac{du_{\mathbf{\Theta}}}{dx}|_{x_i} - 6u_{\mathbf{\Theta}}(x_i)\Big)\times \\ & \frac{\partial\Big(\frac{d^2u_{\mathbf{\Theta}}}{dx^2}|_{x_i} + \frac{du_{\mathbf{\Theta}}}{dx}|_{x_i} - 6u_{\mathbf{\Theta}}(x_i)\Big)}{\partial\theta} \end{aligned} \tag{B.14}$$

For notational simplicity, let $u^{(1)}_{\mathbf{\Theta}}$ denote $\frac{du_{\mathbf{\Theta}}}{dx}$, and $u^{(2)}_{\mathbf{\Theta}}$ denote $\frac{d^2u_{\mathbf{\Theta}}}{dx^2}$. For simplicity, start off with the weights in the last (linear) layer, i.e., consider θ as one of the weights $\mathbf{W}^p_{D+1}$.

$$\begin{aligned} u_{\mathbf{\Theta}} &= \sum_{p=1}^{P}\mathbf{W}^p_{D+1}\mathbf{z}^p_D + \mathbf{b}_{D+1} \\ \frac{\partial u_{\mathbf{\Theta}}}{\partial\mathbf{W}^p_{D+1}} =& \mathbf{z}^p_D \end{aligned} \tag{B.15}$$

$$
\begin{aligned}
u_{\mathbf{\Theta}}^{(1)} &= \sum_{p=1}^{P} \mathbf{W}_{D+1}^{p} \mathbf{G}_{D}^{p} \\
\frac{\partial u_{\mathbf{\Theta}}^{(1)}}{\partial \mathbf{W}_{D+1}^{p}} &= \mathbf{G}_{D}^{p}
\end{aligned}
\tag{B.16}
$$

$$
\begin{aligned}
u_{\mathbf{\Theta}}^{(2)} &= \sum_{p=1}^{P} \mathbf{W}_{D+1}^{p} \mathbf{E}_{D}^{p} \\
\frac{\partial u_{\mathbf{\Theta}}^{(2)}}{\partial \mathbf{W}_{D+1}^{p}} &= \mathbf{E}_{D}^{p}
\end{aligned}
\tag{B.17}
$$

Further, consider θ as one of the weights $\mathbf{W}_{D}^{qp}$.

$$
\begin{aligned}
\frac{\partial u_{\mathbf{\Theta}}}{\partial \mathbf{W}_{D}^{qp}} &= \frac{\partial u_{\mathbf{\Theta}}}{\partial \mathbf{z}_{D}^{p}} . \frac{\partial \mathbf{z}_{D}^{p}}{\partial \mathbf{L}_{D}^{p}} . \frac{\partial \mathbf{L}_{D}^{p}}{\partial \mathbf{W}_{D}^{qp}} \\
\implies \frac{\partial u_{\mathbf{\Theta}}}{\partial \mathbf{W}_{D}^{qp}} &= \mathbf{W}_{D+1}^{p} \sigma^{'}(\mathbf{L}_{D}^{p}) \mathbf{Z}_{D-1}^{q}
\end{aligned}
\tag{B.18}
$$

$$
\frac{\partial u_{\mathbf{\Theta}}^{(1)}}{\partial \mathbf{W}_{D}^{qp}} = \frac{\partial T_1}{\partial \mathbf{G}_{D}^{p}} . \frac{\partial \mathbf{G}_{D}^{p}}{\partial \mathbf{W}_{D}^{qp}} \tag{B.19}
$$

$$
\begin{aligned}
\mathbf{G}_D^p &= \sigma^{'}(\mathbf{L}_D^p)\mathbf{H}_D^p \\
\implies \mathbf{G}_D^p &= \sigma^{'}(\sum_{q_1=1}^{P}\mathbf{W}_D^{q_1 p}\mathbf{z}_{D-1}^{q_1} + \mathbf{b}_D^p).(\sum_{q_1=1}^{P}\mathbf{W}_D^{q_1 p}\mathbf{G}_{D-1}^{q_1}) \\
\implies \frac{\partial \mathbf{G}_D^p}{\partial \mathbf{W}_D^{qp}} &= \sigma^{''}(\mathbf{L}_D^p)\mathbf{z}_{D-1}^q\mathbf{H}_D^p + \sigma^{'}(\mathbf{L}_D^p)\mathbf{G}_{D-1}^q \\
\frac{\partial u_{\mathbf{\Theta}}^{(1)}}{\partial \mathbf{G}_D^p} &= \mathbf{W}_{D+1}^p
\end{aligned}
\tag{B.20}
$$

$$
\frac{\partial u_{\mathbf{\Theta}}^{(1)}}{\partial \mathbf{W}_D^{qp}} = \mathbf{W}_{D+1}^p\Big(\sigma^{''}(\mathbf{L}_D^p)\mathbf{z}_{D-1}^q\mathbf{H}_D^p + \sigma^{'}(\mathbf{L}_D^p)\mathbf{G}_{D-1}^q\Big) \tag{B.21}
$$

Second order

$$
\begin{aligned}
\frac{\partial u_{\mathbf{\Theta}}^{(2)}}{\partial \mathbf{W}_D^{qp}} &= \frac{\partial u_{\mathbf{\Theta}}^{(2)}}{\partial \mathbf{E}_D^p}.\frac{\partial \mathbf{E}_D^p}{\partial \mathbf{W}_D^{qp}} \\
\frac{\partial \mathbf{E}_D^p}{\partial \mathbf{W}_D^{qp}} &= \frac{\partial \mathbf{C}_D^p}{\partial \mathbf{W}_D^{qp}} + \frac{\partial\Big(\sigma^{'}(\mathbf{L}_D^p)\mathbf{F}_D^p\Big)}{\partial \mathbf{W}_D^{qp}}
\end{aligned}
\tag{B.22}
$$

$$
\begin{aligned}
\frac{\partial \mathbf{C}_D^p}{\partial \mathbf{W}_D^{qp}} &= \frac{\partial \Big(\sigma^{''}(\sum_{q_1=1}^{P} \mathbf{W}_D^{q_1 p} \mathbf{z}_{D-1}^{q_1} + \mathbf{b}_D^p)(\mathbf{H}_D^p)^2 \Big)}{\partial \mathbf{W}_D^{qp}} \\
\frac{\partial \mathbf{C}_D^p}{\partial \mathbf{W}_D^{qp}} &= \frac{\partial \Big(\sigma^{''}(\sum_{q_1=1}^{P} \mathbf{W}_D^{q_1 p} \mathbf{z}_{D-1}^{q_1} + \mathbf{b}_D^p)(\sum_{q_1=1}^{P} \mathbf{W}_D^{q_1 p} \mathbf{G}_{D-1}^{q_1})^2 \Big)}{\partial \mathbf{W}_D^{qp}} \\
\frac{\partial \mathbf{C}_D^p}{\partial \mathbf{W}_D^{qp}} =& \frac{\partial \Big(\sigma^{''}(\sum_{q_1=1}^{P} \mathbf{W}_D^{q_1 p} \mathbf{z}_{D-1}^{q_1} + \mathbf{b}_D^p) \Big)}{\partial \mathbf{W}_D^{qp}} (\mathbf{H}_D^p)^2 \\
&+ \frac{(\sum_{q_1=1}^{P} \mathbf{W}_D^{q_1 p} \mathbf{G}_{D-1}^{q_1})^2}{\partial \mathbf{W}_D^{qp}} \sigma^{''}(\sum_{q_1=1}^{P} \mathbf{W}_D^{q_1 p} \mathbf{z}_{D-1}^{q_1} + \mathbf{b}_D^p) \\
\frac{\partial \mathbf{C}_D^p}{\partial \mathbf{W}_D^{qp}} =& \sigma^{'''}(\mathbf{L}_D^p) \mathbf{z}_{D-1}^q (\mathbf{H}_D^p)^2 + 2\sigma^{''}(\mathbf{L}_D^p) \mathbf{H}_D^p G_{D-1}^q
\end{aligned}
\tag{B.23}
$$

$$
\frac{\partial \Big(\sigma^{'}(\mathbf{L}_D^p) \mathbf{F}_D^p \Big)}{\partial \mathbf{W}_D^{qp}} = \sigma^{''}(\mathbf{L}_D^p) \mathbf{z}_{D-1}^q \mathbf{F}_D^p + \sigma^{'}(\mathbf{L}_D^p) \mathbf{E}_{D-1}^q \tag{B.24}
$$

$$
\begin{aligned}
\frac{\partial u_{\mathbf{\Theta}}^{(2)}}{\partial \mathbf{W}_D^{qp}} &= \mathbf{W}_{D+1}^p \frac{\partial \mathbf{E}_D^p}{\partial \mathbf{W}_D^{qp}} \\
\implies \frac{\partial u_{\mathbf{\Theta}}^{(2)}}{\partial \mathbf{W}_D^{qp}} &= \mathbf{W}_{D+1}^p \Big(\sigma^{'''}(\mathbf{L}_D^p) \mathbf{z}_{D-1}^q (\mathbf{H}_D^p)^2 \\
&\quad + \sigma^{''}(\mathbf{L}_D^p)(\mathbf{z}_{D-1}^q \mathbf{F}_D^p + 2\mathbf{G}_{D-1}^q \mathbf{H}_D^p) \\
&\quad + \sigma^{'}(\mathbf{L}_D^p) \mathbf{E}_{D-1}^q \Big)
\end{aligned}
\tag{B.25}
$$

As shown above, the gradient of $u_{\mathbf{\Theta}}^{(2)}$ has an extra $\sigma^{'}$ term in addition to the $\sigma^{''}$ and $\sigma^{'''}$ for a multi-layer NN.

B.5 Hyperparameter Tuning

The parameters for each activation function are tuned with grid search using a set of appropriate values at a certain level (*low* or *medium* or *high*), and the best combination of parameters is chosen. Except for the NN for modeling nonlinear discontinuous function in Section 3.1, all the PINNs use the L-BFGS optimizer. Adam optimizer[68] is used in Section 3.1.

Table B.1: Tuned learning rates for all the studies in Section 3.

Sections	Regression	First-order ODE	Second-order ODE
Stan	0.008	0.5	0.25
tanh	0.008	0.25	0.25
N-LAAF	0.08	0.5	0.5
Swish	0.08	1	0.05
Rowdy	0.008	0.05	0.25

Table B.2: Tuned parameters for all the studies in Section 3. β_{init} denotes the initial β value for all the neurons. RT denotes number of terms in the Rowdy activation function.

	Stan	tanh(AF)	Rowdy
Regression Problem	$\beta_{init} = 0.25$	$n = 1$	$n = 10$, $RT = 3$
First-order ODE	$\beta_{init} = 1$	$n = 3$	$n = 8$, $RT = 3$
Second-order ODE	$\beta_{init} = 1$	$n = 3$	$n = 5$, $RT = 3$

For the Adam optimizer and the L-BFGS optimizer, the search space of learning rates for all the activation functions are $\{8e{-}6, 8e{-}5, 8e{-}4, 8e{-}3, 8e{-}2\}$ and $\{1, 0.5, 0.25, 0.1, 0.05\}$, respectively. The adaptive activation function[60] is tuned on a 2D grid with $n \in \{1, 3, 5, 8, 10\}$. The proposed Stan function is tuned with constant β initialization $\beta_{init} \in \{0, 0.25, 0.5, 1\}$. The Rowdy function[63] is tuned on a 3D grid with the corresponding learning rates, $n \in \{1, 3, 5, 8, 10\}$ and the number of Rowdy terms $RT \in \{3, 7, 9\}$. The tanh and Swish[114] do not have any tuning parameters apart from the learning rates.

B.6 More Results on Numerical Studies

The section shows the predictions at *low* (Figure B.3) and *medium* (Figure B.4) levels for the regression problem, first-order ODE, and second-order ODE. Table B.3 shows the average mean squared error of predictions for all the numerical studies.

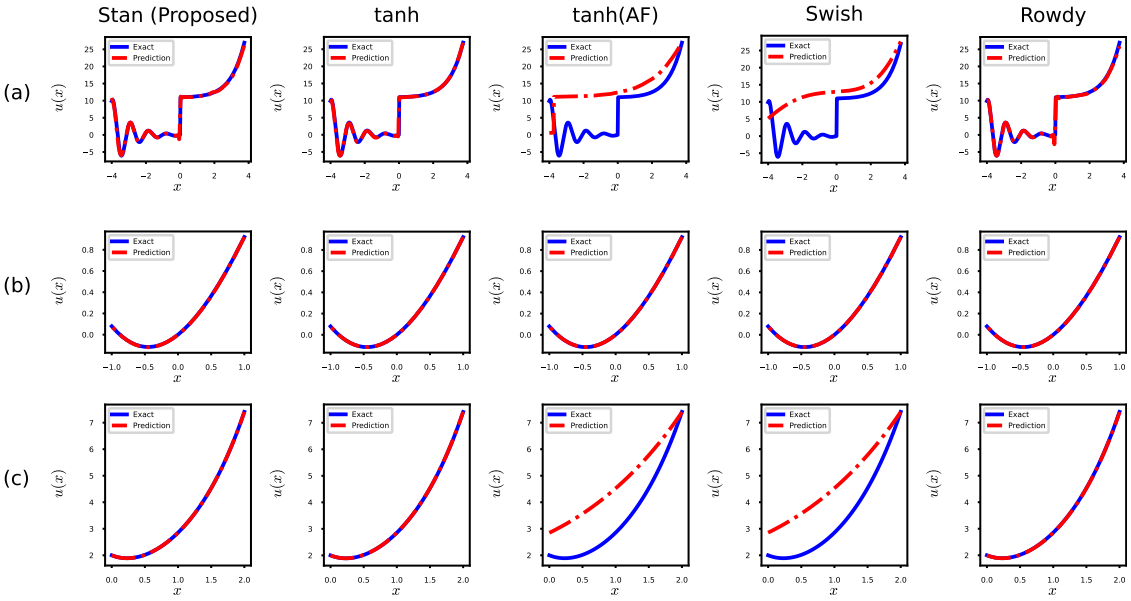


Figure B.3: Average predictions at the *low* level of the trained NNs/PINNs with various activation functions in comparison with the exact function for (a) regression problem (b) first-order differential equation, and (c) second-order differential equation.

Table B.3: Average performance in predicting the solution for the regression problem, first-order differential equation, and second-order differential equation in terms of MSE.

	Regression			First-order ODE			Second-order ODE		
	L	M	H	L	M	H	L	M	H
Stan (Proposed)	**0.24**	**32.32**	**0.04**	**0.00**	**0.00**	**2e3**	**0.00**	**0.00**	**2e3**
tanh	0.34	169.13	2e4	0.00	476.40	4e6	0.00	65.84	5e5
tanh (AF) [60, 61]	33.43	2e3	6e4	0.00	3e3	5e6	0.00	349.44	1e4
Swish [114]	19.24	1e3	2e4	0.00	0.00	2e3	0.00	153.49	1e4
Rowdy [63]	0.51	4.73	745.80	0.00	0.04	1e6	1.37	60.73	1e4

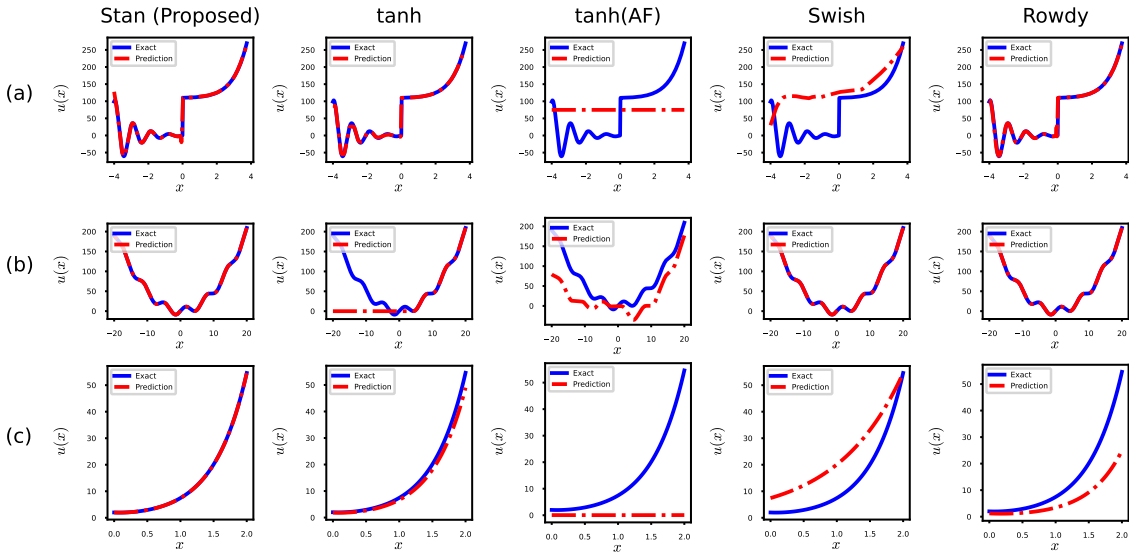


Figure B.4: Average predictions at *medium* level of the trained NNs/PINNs with various activation functions in comparison with the exact function for (a) regression problem (b) first-order differential equation, and (c) second-order differential equation.

B.7 Details of Case Studies

The hyperparameters are given in Table B.4. Figure B.5 shows the training convergence over the epochs. The mean squared error of predictions are provided in Table B.5.

Table B.4: Hyperparameters for all the case studies in Section 4. KG denotes Klein Gordon Equation. NLHT denotes Non-linear Heat Transfer. MSD denotes Mass Spring Damper.

		KG	NLHT	MSD
	Width	50	50	50
	Hidden Layers	4	3	3
Stan	(lr,β_{init})	(1,1)	(0.25,0.5)	(0.25,1)
tanh	lr	1	1	0.25
tanh (AF)	(lr,n)	(1,1)	(0.1,1)	(0.5,1)
Swish	lr	1	0.05	0.05
Rowdy	(lr,n,RT)	(1,5,6)	(0.1,1,6)	(0.25,5,2)

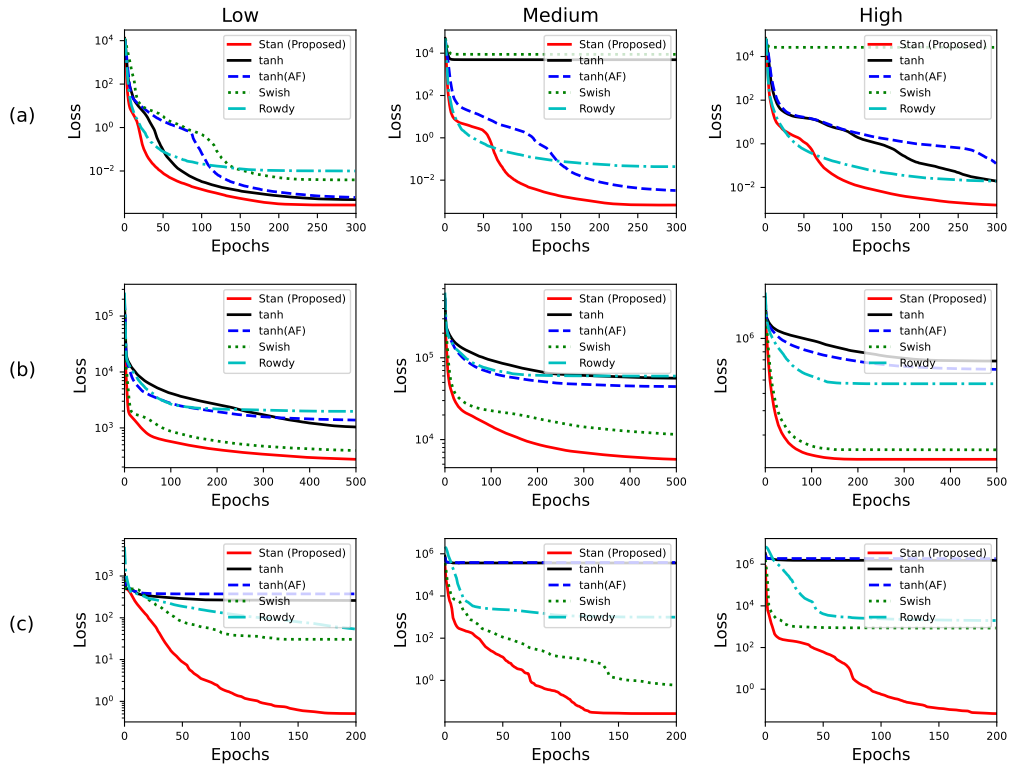


Figure B.5: Training loss convergence plots averaged over 10 different initialization of weights and biases of the NNs/PINNs with various activation functions for (a) Klein-Gordon equation (b) Non-linear Heat Transfer, and (c) Mass-Spring-Damper at the corresponding three levels considered in the main text.

Table B.5: Average performance in predicting the entire solution for the case studies in terms of MSE.

	Klein-Gordon Equation			Non-linear Heat Transfer			Mass-Spring-Damper		
	L	M	H	L	M	H	L	M	H
Stan (Proposed)	**0.00**	**0.00**	**0.00**	**24.55**	**540.21**	**1e4**	**0.19**	**0.00**	**0.01**
tanh	0.00	0.44	0.01	151.97	1e4	1e5	245.30	3e5	1e6
tanh (AF) [60, 61]	0.00	0.00	0.01	258.67	8e3	9e4	351.70	3e5	1e6
Swish [114]	0.00	3.41	1.13	41.44	1,e3	1e4	19.14	0.09	158.61
Rowdy [63]	0.0	0.00	0.00	331.42	1e4	6e4	25.97	137.55	308.06

B.7.1 Klein-Gordon Equation

Solving the Klein-Gordon includes choosing 6000 points for the set $\mathcal{U}_1$, 5000 points for the set $\mathcal{U}_2$ and 10,000 points for the set $\mathcal{F}$. Eq.(B.26) gives the loss function.

$$
\begin{aligned}
J(\tilde{\mathbf{\Theta}}) = \frac{1}{10000}\sum_{i=1}^{10000}\Bigg|&\frac{\partial^2 u_{\tilde{\mathbf{\Theta}}}}{\partial t^2}|_{(x_i,t_i)} - \frac{\partial^2 u_{\tilde{\mathbf{\Theta}}}}{\partial x^2}|_{(x_i,t_i)} + u^3_{\tilde{\mathbf{\Theta}}}(x_i,t_i) \\
&+ 25\pi^2 x_i \cos(5\pi t_i) - 6x_i^3 t_i + 6x_i t_i^3 \\
&- (x_i cos(5\pi t_i) + (x_i t_i)^3)^3\Bigg|^2 \\
+\frac{1}{2500}\sum_{i=1}^{2500}&|u_{\tilde{\mathbf{\Theta}}}(x_{max},t_i) - h(x_{max},t_i)|^2 \\
+\frac{1}{2500}\sum_{i=1}^{2500}&|u_{\tilde{\mathbf{\Theta}}}(x_{min},t_i) - h(x_{min},t_i)|^2 \\
+\frac{1}{1000}\sum_{i=1}^{1000}&|u_{\tilde{\mathbf{\Theta}}}(x_i,0)|^2 \\
+\frac{1}{3500}\sum_{i=1}^{3500}&\left|\frac{\partial u_{\tilde{\mathbf{\Theta}}}}{\partial y}|_{(x,y,t)\in\mathcal{U}_2}\right|^2 .
\end{aligned}
\tag{B.26}
$$

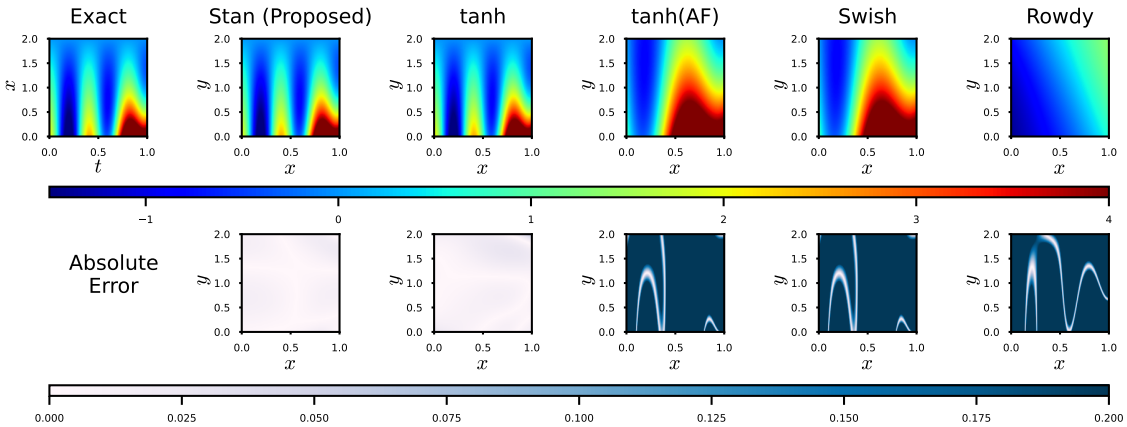


Figure B.6: Representative prediction of Klein-Gordon Equation at the *medium* level.

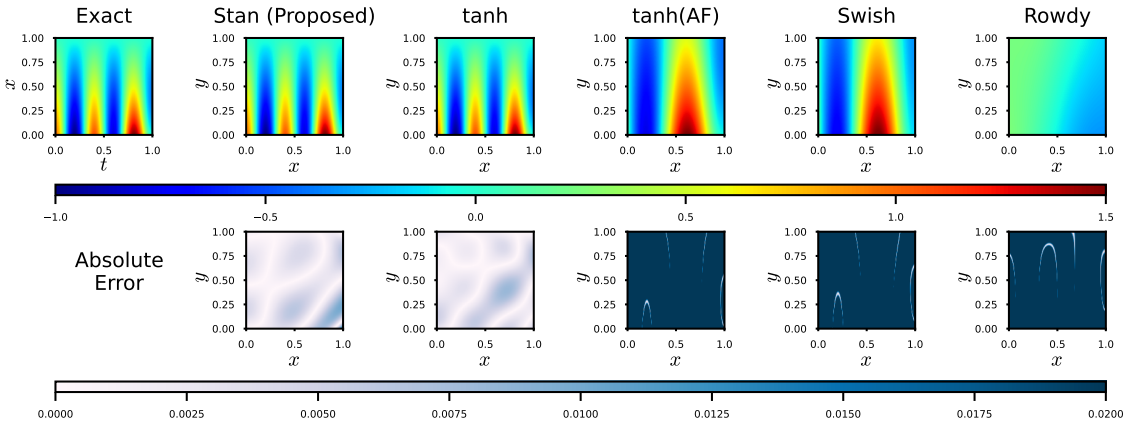


Figure B.7: Representative prediction of Klein-Gordon Equation at the *low* level.

x_{min} and x_{max} are determined by definition of Ω. The predictions at the *medium* and the *low* levels are shown in Figure B.6 and Figure B.7 respectively.

B.7.2 Non-linear Heat Transfer (NLHT) Problem

The constant parameters used in the heat transfer equation (Eq. (28)) is given in Table B.6. The loss function is given as

$$
\begin{aligned}
J(\tilde{\mathbf{\Theta}}) = \frac{1}{10000}\sum_{i=1}^{10000} \Bigg| & \rho C_p t_c \frac{\partial u_{\tilde{\mathbf{\Theta}}}}{\partial t}|_{(x_f^i, y_f^i, t_f^i)} \\
& - k t_c \left(\frac{\partial^2 u_{\tilde{\mathbf{\Theta}}}}{\partial x^2}|_{(x_f^i, y_f^i, t_f^i)} + \frac{\partial^2 u_{\tilde{\mathbf{\Theta}}}}{\partial y^2}|_{(x_f^i, y_f^i, t_f^i)} \right) \\
& + 2h_c(u_{\tilde{\mathbf{\Theta}}}(x_f^i, y_f^i, t_f^i) - u_a) \\
& + 2\epsilon\sigma(u^4_{\tilde{\mathbf{\Theta}}}(x_f^i, y_f^i, t_f^i) - u_a^4) \Bigg|^2 \\
+ \frac{1}{5000}\sum_{i=1}^{5000} & \left| u_{\tilde{\mathbf{\Theta}}}(x_{u_1}^i, 0, t_{u_1}^i) - u_0 \right|^2 \\
+ \frac{1}{5000}\sum_{i=1}^{5000} & \left| u_{\tilde{\mathbf{\Theta}}}(x_{u_1}^i, y_{u_1}^i, 0) - 300 \right|^2 \\
+ \frac{1}{3500}\sum_{i=1}^{3500} & \left| \frac{\partial u_{\tilde{\mathbf{\Theta}}}}{\partial y}|_{(x,y,t)\in\mathcal{U}_2} \right|^2 .
\end{aligned}
\tag{B.27}
$$

A representative prediction at the *low* and the *high* levels are provided in Figure B.8 and Figure B.9 respectively.

Table B.6: The constant parameters in Eq.(28).

PDE Parameter	Value
Convection Coefficient (h_c)	$1W/(m^2K)$
Emissivity (ϵ)	0.5
Material Density (ρ)	$8960kg/m^3$
Thermal Conductivity (k)	$400W/mK$
Specific Heat (C_p)	$386J/kgK$
Thickness of the plate (t_c)	$0.01m$
Stefan-Boltzmann Constant (σ)	$5.67 \times 10^{-8}W/m^2K^4$

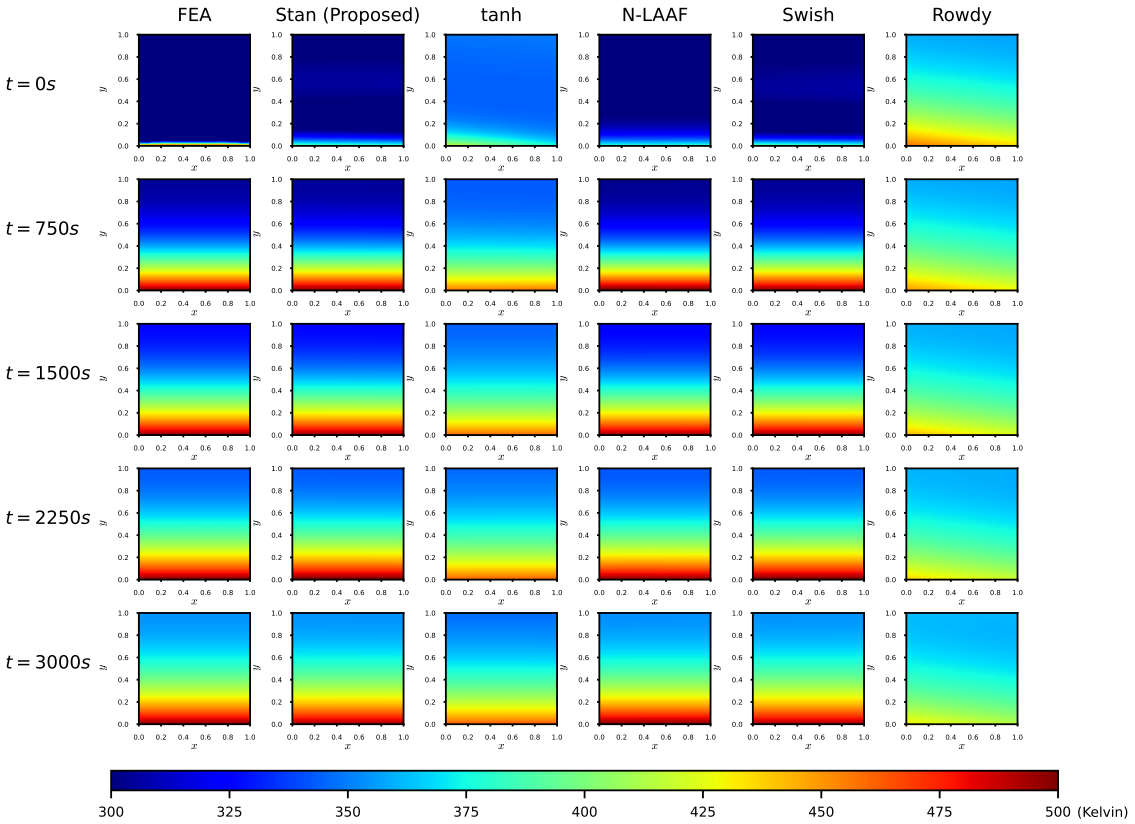


Figure B.8: Comparison of the FEA solution of thermal history in a thin plate with PINN solutions of various activation functions at few representative time steps for the *low* level.

B.7.3 System Identification of Mass Spring Damper

A representative reconstruction of solution at the *medium* level and at the *low* level are given in Figure B.10 and Figure B.11.

The convergence of the predicted differential equation parameters m,c, and k is provided in Figures B.12 and B.13.

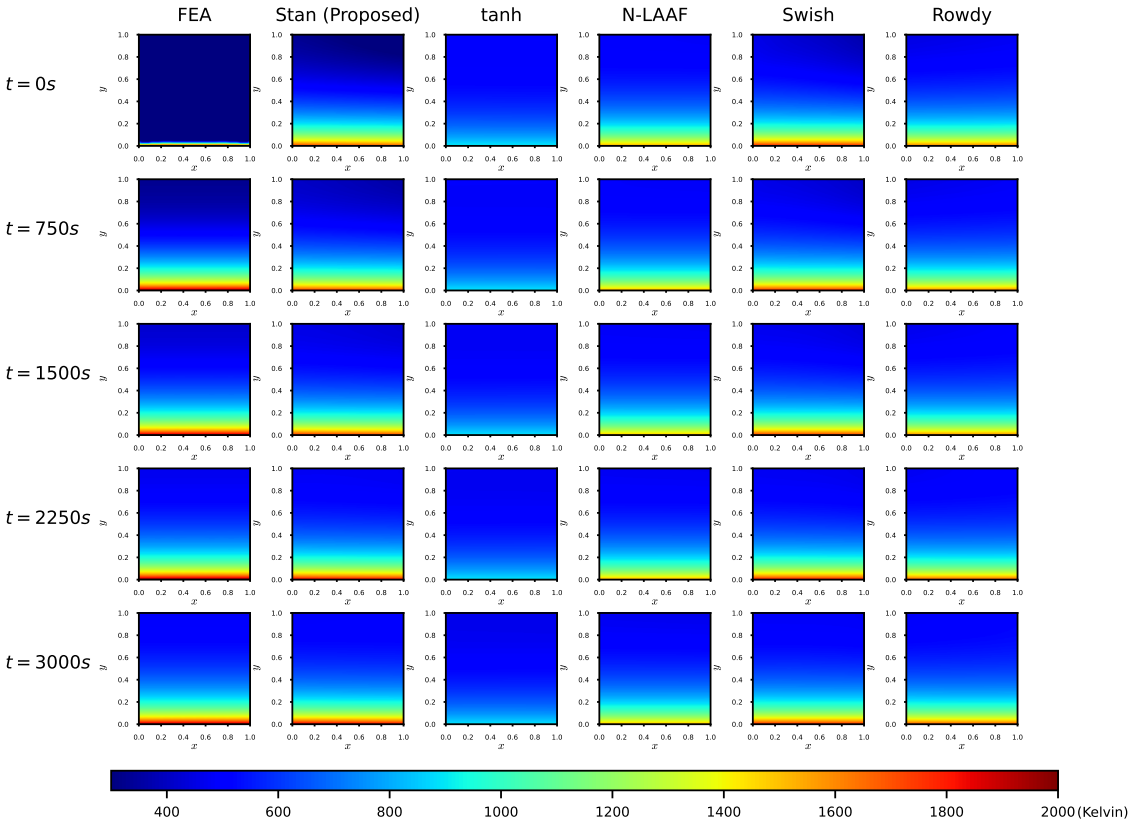


Figure B.9: Comparison of the FEA solution of thermal history in a thin plate with PINN solutions of various activation functions at few representative time steps for the *high* level.

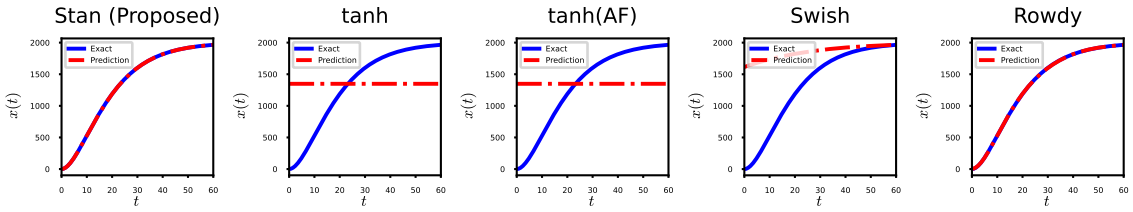


Figure B.10: Representative reconstruction of solution using the data from a Mass-Spring-Damper system and governing equation at the *medium* level.

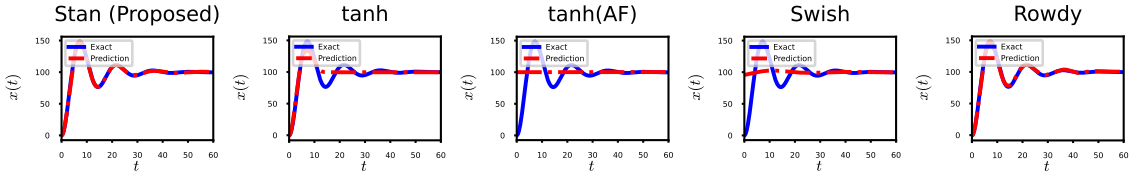


Figure B.11: Representative reconstruction of solution using the data from a Mass-Spring-Damper system and governing equation at the *low* level.

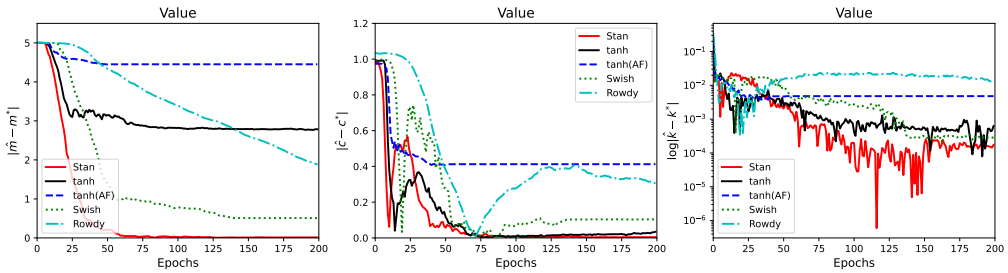


Figure B.12: Convergence of predicting the values of m,c, and k for the *low* level Mass-Spring-Damper equation.

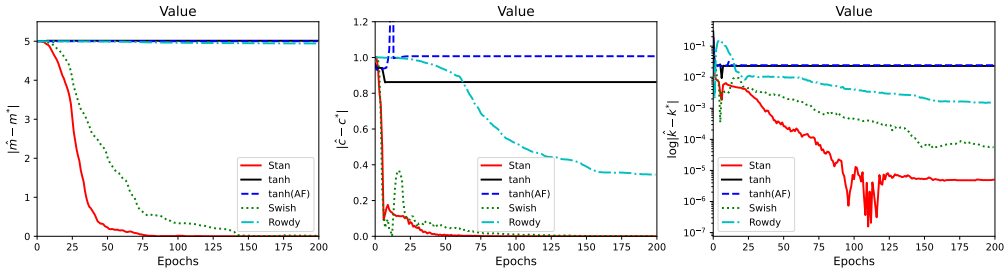


Figure B.13: Convergence of predicting the values of m,c, and k for the *medium* level Mass-Spring-Damper equation.

B.8 Additional Case Studies

B.8.1 Inverse Heat Transfer in a Rod

Figure B.14 shows a rod of unknown material, with ends maintained at a constant temperature of 0 units. The rod is considered at a temperature of u_0 units and starts cooling down at $t = 0$. Assuming we have the potentially noisy temperature data at sufficient points in the rod throughout the cooling process, the aim is to identify the material of the rod. To identify the material, ignoring all the thermal effects except conduction, it is sufficient to figure out the characteristic thermal diffusivity of the rod κ.

We can model the heat loss with a simple PDE system as in Eq.(B.28), considering the rod

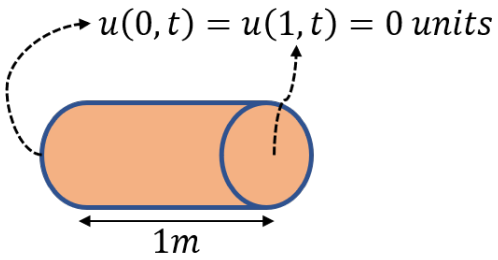


Figure B.14: Problem setting to identify the material property, κ, given data at various points over time.

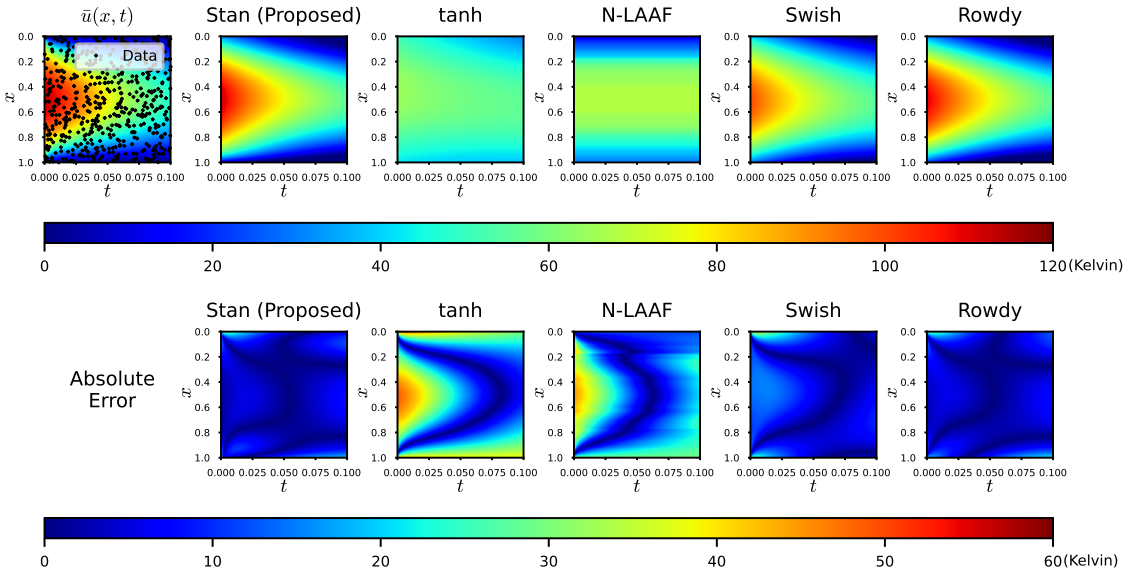


Figure B.15: The data provided to the PINN is marked in the backdrop of the visualization of the closed form approximation (top left). The average reconstructed visualizations from the data points by including the physics-information (as PDE) is shown on the top for all the activation functions. The corresponding absolute error between the reconstructions and the closed form approximations is visualized below. The absolute error between the Stan activation function and the Rowdy activation function look similar.

as single dimensional.

$$\begin{aligned} \texttt{PDE}: &\ \frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2},\ 0 < x < 1, \\ \texttt{DBC1}: &\ u(x,0) = u_0,\ 0 < x < 1, \\ \texttt{DBC2}: &\ u(0,t) = u(1,t) = 0,\ t > 0, \end{aligned} \tag{B.28}$$

The goal is to figure out the material property, κ. We consider a particular instance of the problem, where the data $u(x,t)$ is available at 5000 spatio-temporal points (data may or may not be available at the boundaries) in the support. A typical NN can approximate the solution $u(x,t)$ with the provided data. To identify the thermal diffusivity κ, the PINNs supplement the NN with the information of the PDE. Specifically, the loss function is,

$$\begin{aligned} J(\tilde{\boldsymbol{\Theta}},\kappa) =& \frac{w_{\mathcal{F}}}{50000}\sum_{i=1}^{50000}\left|\frac{\partial u_{\tilde{\boldsymbol{\Theta}},\kappa}}{\partial t}|_{(x_f^i,t_f^i)} - \kappa\frac{\partial^2 u_{\tilde{\boldsymbol{\Theta}},\kappa}}{\partial x^2}|_{(x_f^i,t_f^i)}\right|^2 \\ &+ \frac{w_u}{5000}\sum_{i=1}^{5000}\left|\bar{u}(x_u^i,t_u^i) - u_{\tilde{\boldsymbol{\Theta}},\kappa}(x_u^i,t_u^i)\right|^2, \end{aligned} \tag{B.29}$$

where the κ is unknown and a trainable parameter in addition to the PINN's parameters $\tilde{\boldsymbol{\Theta}}$. The collocation points for minimizing the `PDE residual loss` are 50,000. $\bar{u}$ denotes the potentially noisy data available at $(x_u^i,t_u^i)\ \forall\ i = \{1,2,3,...,5000\}$. Note that we ignore the non-dimensionalization trick for showing the versatility of the proposed activation function.

To simulate this case, the thermal diffusivity $\kappa^* = 1$ is considered the **ground truth**. By choosing $\kappa^* = 1$, the rod cools down sufficiently to reach equilibrium within $t = 0.1s$ for $u_0 = 50units$. For this particular case, the analytical solution[21] is an infinite series,

$$u(x,t) = \frac{4u_0}{\pi}\sum_{i=1}^{\infty}\frac{\sin\Big((2i-1)\pi x\Big)}{2i-1}\exp\Big(-(2i-1)^2\pi^2 t\Big). \tag{B.30}$$

Choosing any finite number of terms from the infinite series results is an approximation $\bar{u}(x,t)$ of the exact function $u(x,t)$. The first 10,000 terms of the series $\bar{u}(x,t)$ is an accurate enough[21] approximation of the function. Training the PINN involves using the thermal measurement data at 5000 randomly chosen spatial-temporal locations from $0 \le x \le 1, 0 \le t \le 0.1$, and their corresponding $\bar{u}(x,t)$. Figure B.15 shows the visualization of the approximate solution and thermal measurement data.

Training the PINN with the loss function in Eq.(B.29) involves considering unit weights for each part of the loss function and the initial assumption of κ as zero. The loss functions for Swish and Rowdy activation functions are as good as the proposed Stan activation function, but the convergence of κ shown in Figure B.16 distinguishes the activation functions. Table B.7 tabulates the mean predicted κ^* and the corresponding absolute errors. The Stan activation function can better estimate the thermal diffusivity κ with the least average absolute error.

In addition to finding κ, PINN can produce a better approximation of the temperature distribution (i.e., solution $u(x,t)$) for the rod as a function of spatial location x and time t, as shown in the prediction and error contours in Figure B.15.

Table B.7: The average predicted values of thermal diffusivity κ using PINNs with various activation functions.

Activation Function	Predicted κ^*	Absolute Error
Stan (Proposed)	**0.9562**	**0.0438**
tanh	-0.5216	1.5216
N-LAAF[60]	0.0594	0.9406
Swish[114]	0.0219	0.9781
Rowdy[63]	0.9350	0.0650

B.8.2 Electrostatic Potential in a Rectangle

Laplace Equation models the Electrostatic potential in a 2D rectangular air-filled space with no charge density. The Laplace equation is popular in the field of science and engineering, particularly prominent in modeling fluid flow and gravitation in addition to electrostatics.

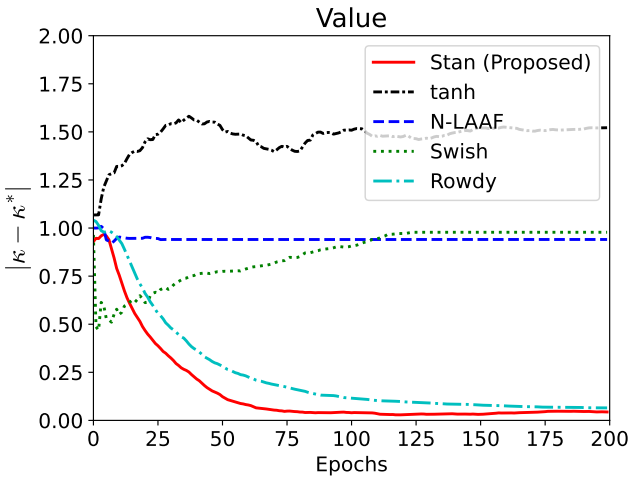


Figure B.16: The convergence of absolute error in predicted the κ value through PINNs with various activation functions.

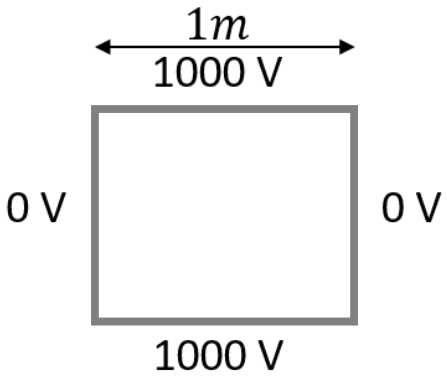


Figure B.17: Problem setting for finding the electrostatic potential within the square using the Laplace equation.

The two-dimensional Laplace Equation in a general form is

$$\begin{aligned} &\nabla^2 u = 0;\ \ x \in [0, a], y \in [0, b] \\ &u(x, 0) = h_1(x); u(x, b) = h_2(x); \\ &u(0, y) = h_3(y); u(a, y) = h_4(y), \end{aligned} \tag{B.31}$$

where u denotes the electrostatic potential and $h_i\ \forall i \in \{1, 2, 3, 4\}$ are known functions. For simplicity, let us assume a unit square space (i.e., $a = 1, b = 1$), and consider constant known

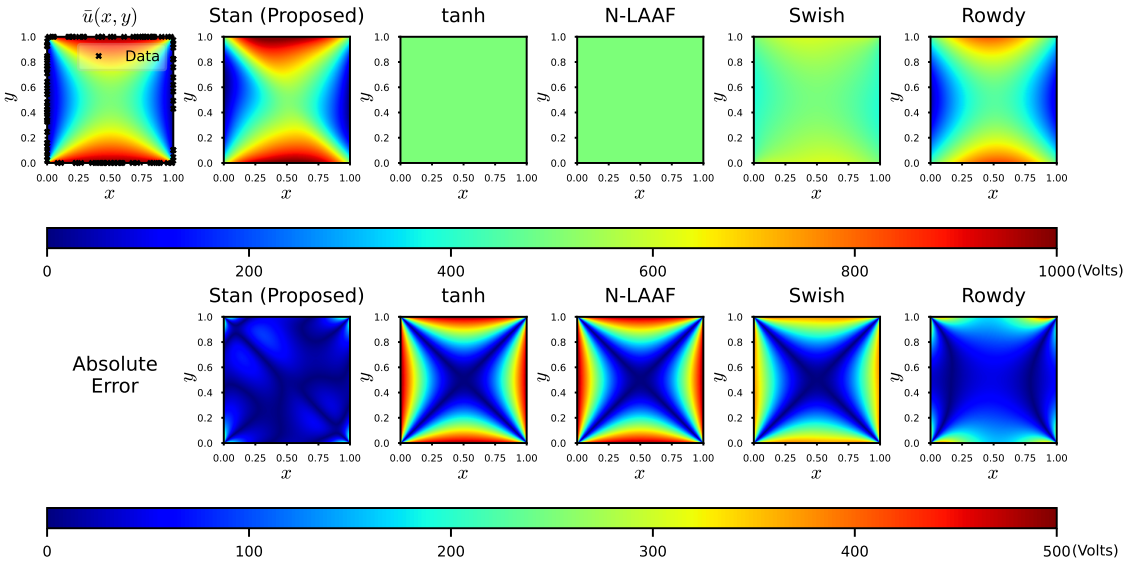


Figure B.18: Visualization of the exact solution (with the data points used for training) in comparison with the solutions predicted from the PINNs with mentioned activation functions averaged over 10 repetitions. The corresponding absolute errors are shown in the figure below. The similarity of the prediction plot and the approximate function plot and the predominant blue color in the absolute error plot shows a better approximation of PINNs with Stan activation function.

functions $h_1(x) = h_2(x) = 1000$ and $h_3(y) = h_4(y) = 0$. Physically, this represents a square with each opposite set of sides having the same electric potential (in Volts) as shown in Figure B.17. The problem is to find the electric potential denoted as $u(x, y)$ in Eq.(B.31) at all the points within the space.

The solution to the Laplace equation using PINN for the particular setting involves choosing a set of 5,000 points for the set $\mathcal{U}_1$. Note that four sides of the square define four parts of the Dirichlet Boundary Condition. So, the set $\mathcal{U}_1$ contains a set of 1250 (5000 divided by 4) randomly chosen points on each side of the square. The set $\mathcal{F}$ consists of 50,000 points sampled by Latin Hypercube Sampling (LHS). Choosing the loss weights as 1, Eq.(B.32) gives the loss function for solving the Laplace equation.

With the assumed conditions, the problem in Eq.(B.31) has a closed-form solution given by an

infinite series[155]. Figure B.18 shows the contour plot of the solution $\bar{u}(x, y)$, by considering the first 100 terms of the series, along with the averaged predicted contours corresponding to each activation function. The Stan function approximates the exact solution remarkably well, and the absolute error contours (in Figure B.18) further exhibit the same. Again, this problem can be non-dimensionalized to make it easier for tanh by avoiding the issue of output scaling.

$$\begin{aligned} J(\tilde{\mathbf{\Theta}}) =& \frac{1}{50000} \sum_{i=1}^{50000} \left| \frac{\partial^2 u_{\tilde{\mathbf{\Theta}}}}{\partial x^2}|_{(x_f^i, y_f^i)} + \frac{\partial^2 u_{\tilde{\mathbf{\Theta}}}}{\partial y^2}|_{(x_f^i, y_f^i)} \right|^2 \\ &+ \frac{1}{1250} \sum_{i=1}^{1250} |u_{\tilde{\mathbf{\Theta}}}(x, 0) - h_1(x)|^2 \\ &+ \frac{1}{1250} \sum_{i=1}^{1250} |u_{\tilde{\mathbf{\Theta}}}(x, 1) - h_2(x)|^2 \\ &+ \frac{1}{1250} \sum_{i=1}^{1250} |u_{\tilde{\mathbf{\Theta}}}(0, y) - h_3(y)|^2 \\ &+ \frac{1}{1250} \sum_{i=1}^{1250} |u_{\tilde{\mathbf{\Theta}}}(1, y) - h_4(y)|^2 . \end{aligned} \tag{B.32}$$

B.9 Literature Problems

In this section, we provide direct comparisons to several problems discussed in literature without the details of implementation.

B.9.1 Burgers' Equation

Burgers' equation with a small value of viscosity parameter is one of the earliest examples that garnered attention towards PINN itself[112]. The equation reads as (directly from

[112]),

$$\begin{aligned} \frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x} - \frac{0.01}{\pi}\frac{\partial^2 u}{\partial x^2} = 0;\ x \in [-1,1], t \in [0,1] \\ u(0,x) = -\sin(\pi x) \\ u(t,-1) = u(t,-1) = 0. \end{aligned} \tag{B.33}$$

The equation is solved at the usual domain [-1,1] (given in Raissi et al.[112]) and an extended domain [-2,2]. Comparisons are shown in Figure B.19 and Table B.8.

Table B.8: Relative Error comparing the Burgers' equation PINN prediction to simulated solution. The proposed activation is better in performance as the domain increases.

Activation Function	RE in [-1,1]	RE in [-2,2]
Stan (Proposed)	**0.0297**	**0.0895**
tanh	0.0252	0.1058

B.9.2 Inverse Navier Stokes Equation

Inverse Navier Stokes is another example from Raissi et al.[112]. We directly use the same form of the equation as,

$$\begin{aligned} u_t + \lambda_1(uu_x + vu_y) = -p_x + \lambda_2(u_{xx} + u_{yy}) \\ v_t + \lambda_1(uv_x + vv_y) = -p_y + \lambda_2(v_{xx} + v_{yy}) \\ u_x + v_y = 0 \end{aligned} \tag{B.34}$$

We use the same trick[112] to combine the velocity components into a single variable as $u = \psi_y$ and $v = -\psi_x$. The problem is to predict the parameters λ_1 and λ_2 and also recover the pressure field from the given velocity field. We use the data directly from the

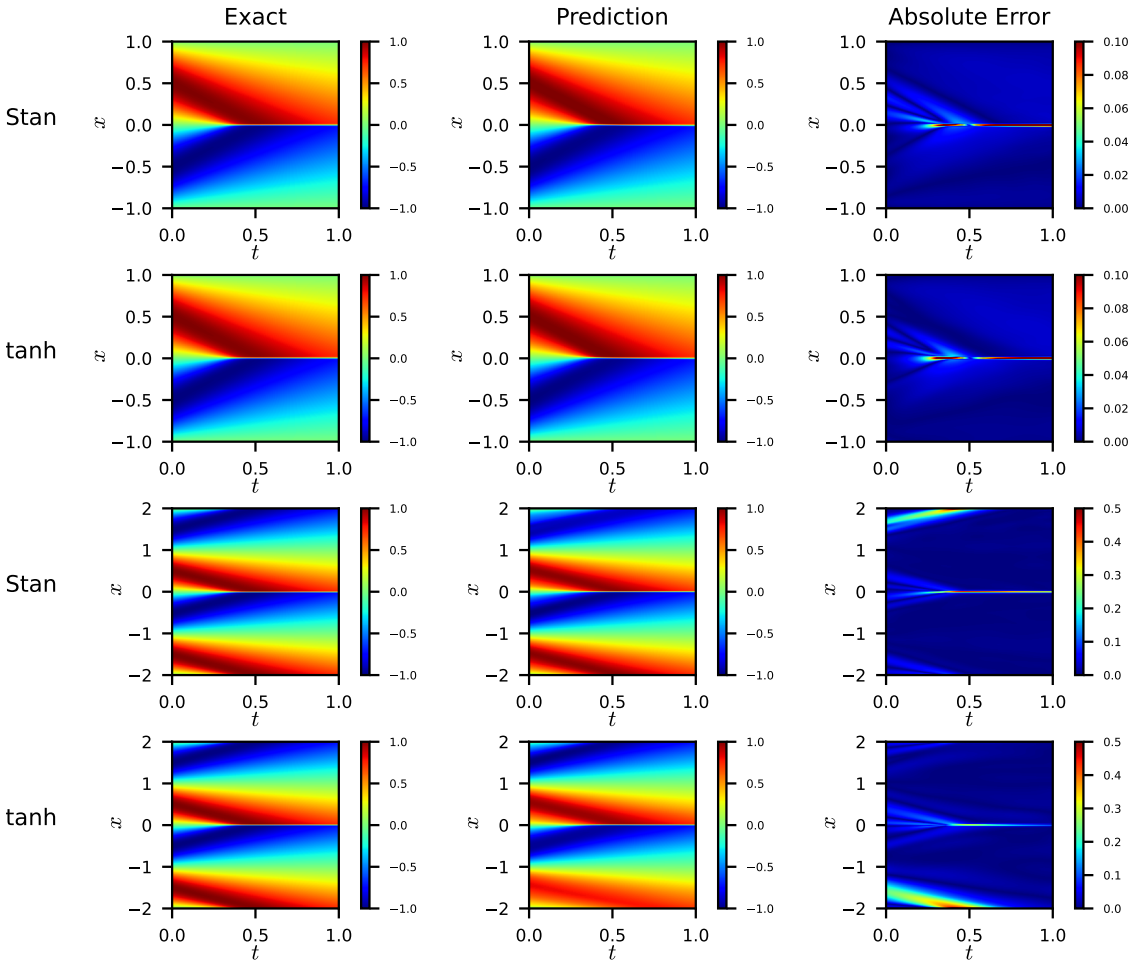


Figure B.19: Comparison on solutions obtained for Burgers' equation with tanh and proposed activation function.

official implementation of Raissi et al. [112] without adding noise. Comparisons are shown in Figure B.20 and Table B.9.

Table B.9: In the inverse Navier Stokes problem, predicted values of λ_1 and λ_2 match for the proposed activation function , tanh, and the true values of 1 and 0.01.

Activation Function	Predicted λ_1	Predicted λ_2
Stan (Proposed)	**0.9997**	**0.0107**
tanh	0.9993	0.0109

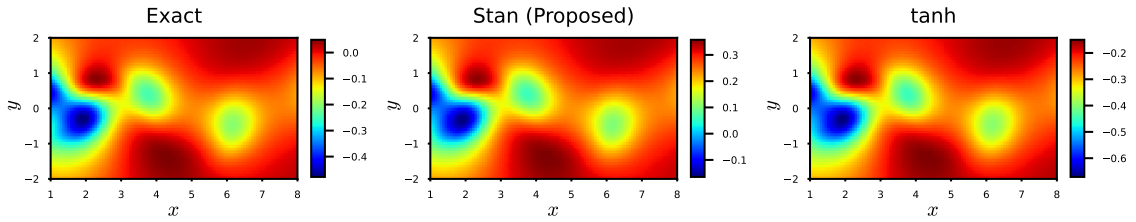


Figure B.20: Comparison on solutions obtained for Navier Stokes equation. The reconstructed pressure fields differ by constant for both the proposed activation function and tanh.

B.9.3 Inverse Sine-Gordon

This problem is taken from the work on adaptive activation function by Jagtap et al.[61]. The equation is skipped here(refer to [61]). The problem is to find the parameters λ_1 and λ_2 (the first 2 parameters in Table 6 of [61]) from clean data. Table B.10 and Figure B.21 summarizes the result.

Table B.10: Predicted values of λ_1 and λ_2 match for the proposed activation function and adaptive activation function in inverse Sine Gordon Equation, and the true values of 4 and 1.

Activation Function	Predicted λ_1	Predicted λ_2
Stan (Proposed)	**3.9994621**	**1.0000045**
tanh(AF)	4.0000596	0.999945

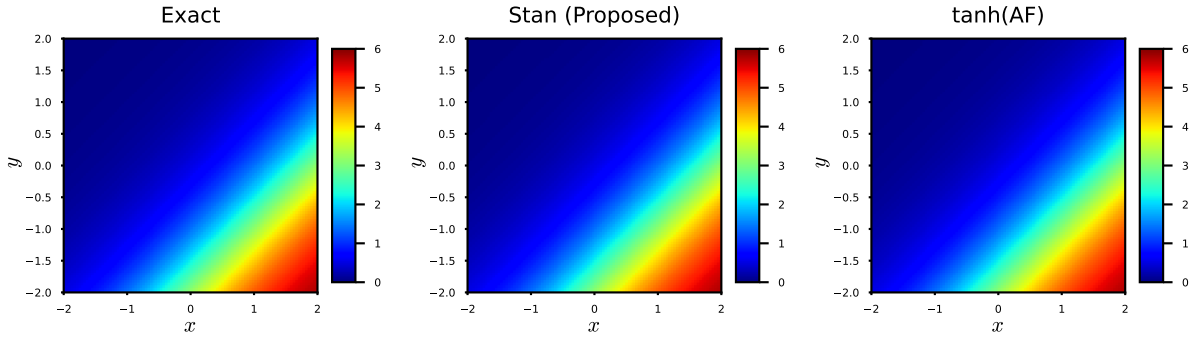


Figure B.21: Comparison on reconstructed solutions obtained for Sine-Gordon Equation. Both the adaptive function and the proposed function can reconstruct the data accurately.

B.9.4 Advection Equation

Solving univariate advection equation is directly taken from the work on self-adaptive PINN (SA-PINN) [91].

$$\begin{aligned} q_t(x,t) + \bar{u} q_x(x,t) = 0, \;\; x \in [0,2], \;\; t \in [0, 0.2] \\ u(0,t) = u(2,t) = 0, t \in [0,2] \\ u(x,0) = g(x), x \in [0,2] \end{aligned} \tag{B.35}$$

Both the tanh function and the Stan function are used with the weighting scheme used by McClenny et al.[91]. The results are summarized in Table B.11.

Table B.11: Relative Error in solving the Advection Equation with tanh and proposed activation function.

Activation Function	RE
Stan (Proposed)	**0.0827**
tanh(AF)	0.1904

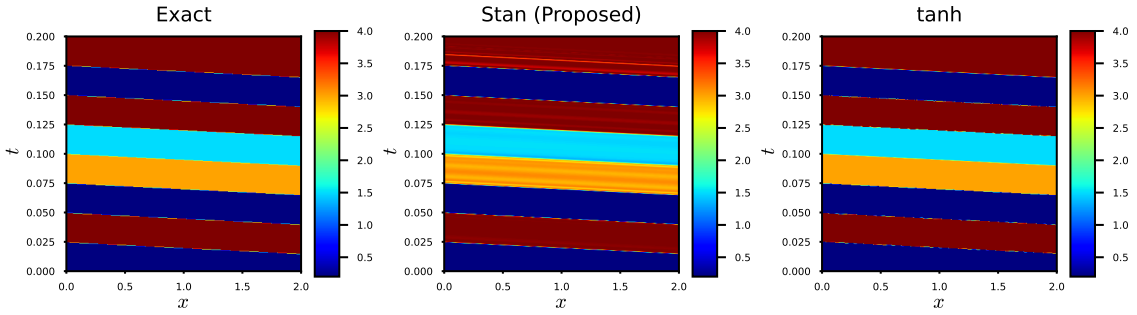


Figure B.22: Comparison on solving Advection Equation with tanh and Stan. Adaptive weighting scheme[91] is used in both of the cases.

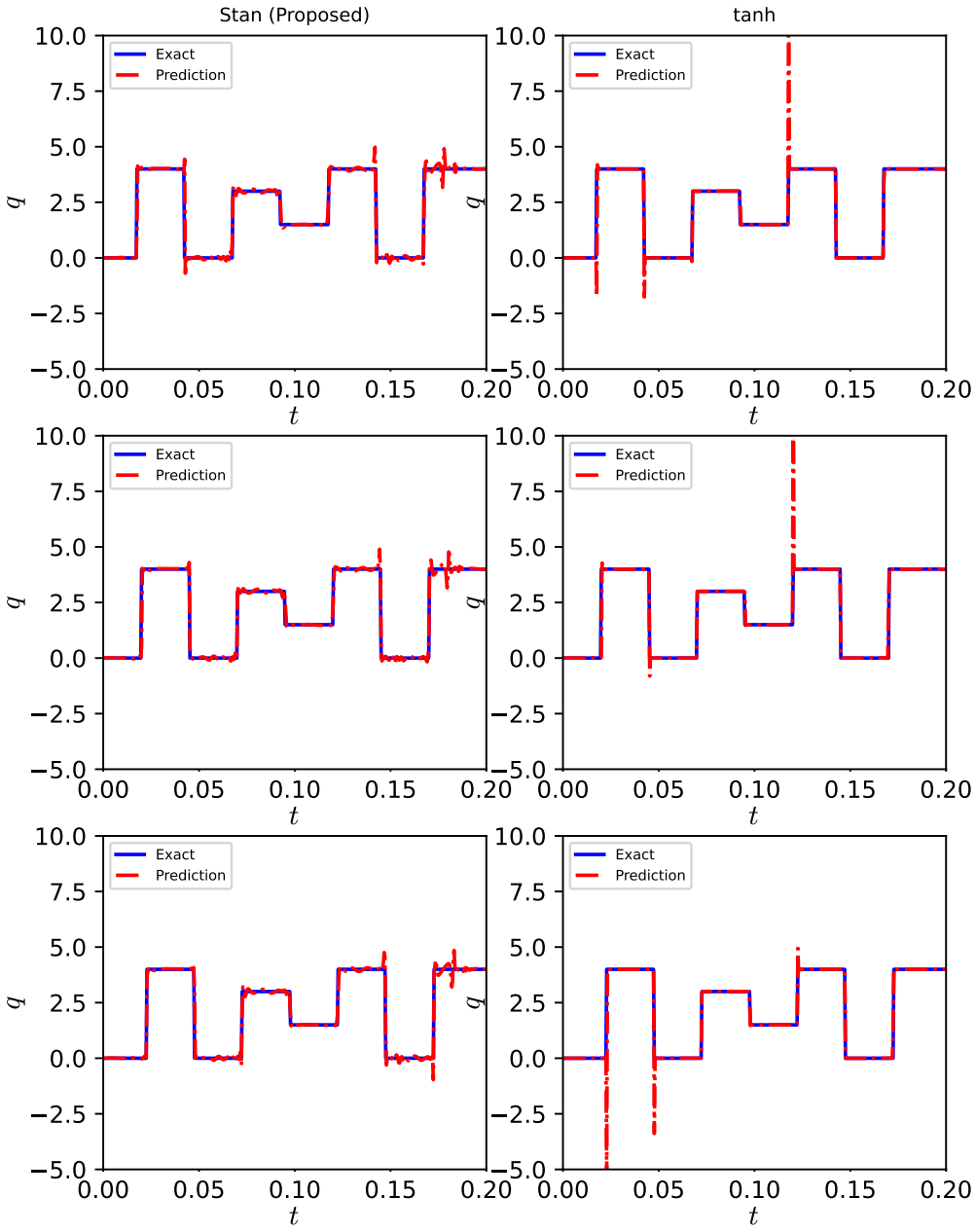


Figure B.23: Comparison on solving Advection Equation with tanh and Stan at a few spatial locations. The RE is lower for the proposed activation function.

B.10 Beta Convergence

This section shows the convergence of the β values for a representative neuron for all the numerical studies and case studies.

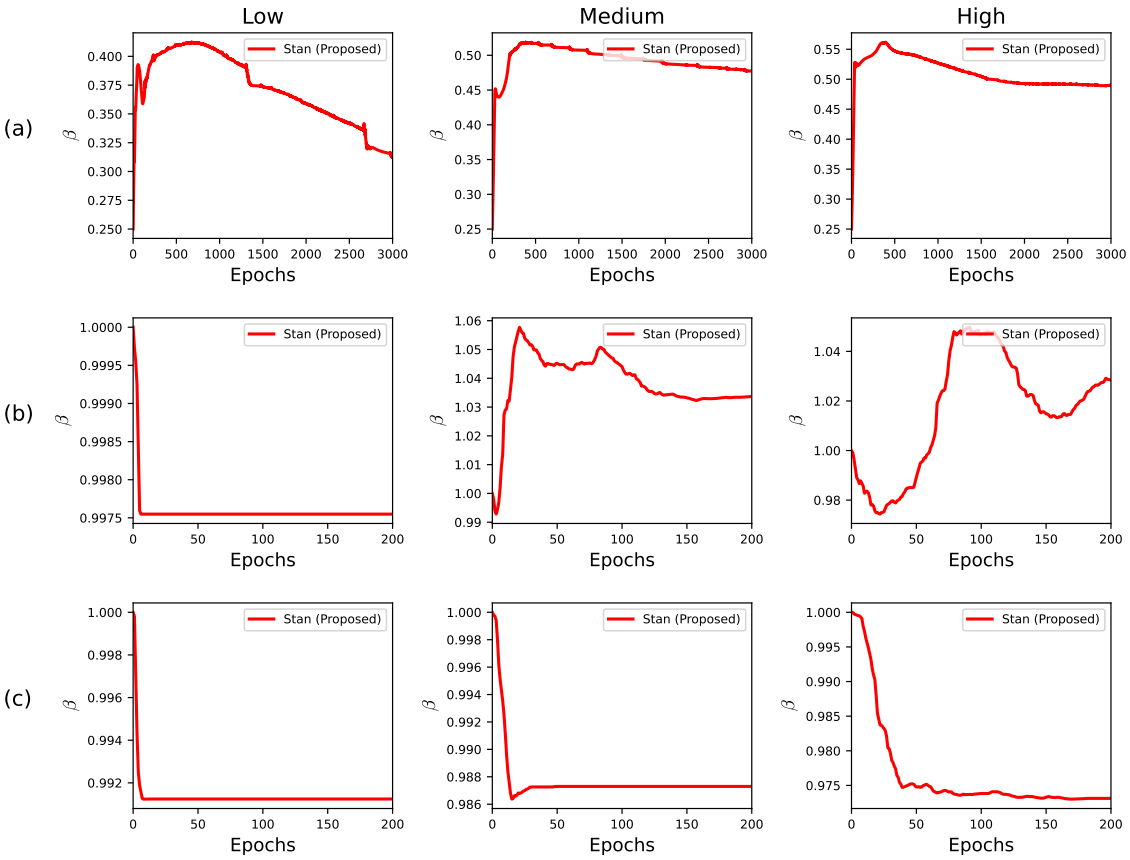


Figure B.24: Convergence of β with the epochs for (a) regression problem (b) first-order differential equation, and (c) second-order differential equation at the related three levels considered in the main text.

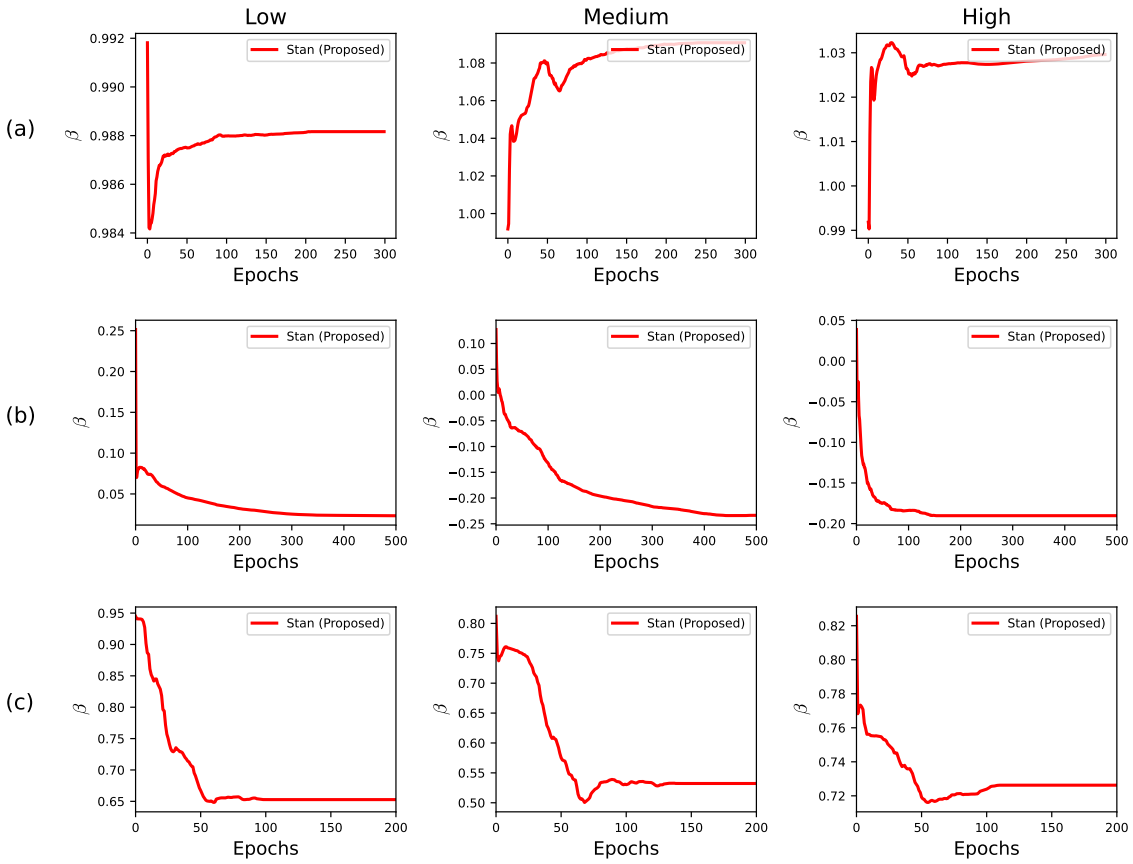


Figure B.25: Convergence of β with the epochs for a) Klein-Gordon equation (b) Non-linear Heat Transfer, and (c) Mass-Spring-Damper at the related three levels considered in the main text.

Appendix C

Appendix for Chapter 5

C.1 Material Properties

The material is taken to be Inconel 625 and the properties were taken to be constant (the values at room temperature).

$$\begin{aligned} \rho &= 8.44 g/cc \\ c_p &= 0.41 J/g - K \\ \kappa &= 9.8 W/m - k \\ r_b &= 1 mm \end{aligned} \tag{C.1}$$

C.2 Implementation Details

PINNs are implemented in Python3 with PyTorch2.1. The architecture is typically 4 layered full connected NN with 50 neurons width. The parameters are optimized with L-BFGS with a learning rate of 0.25.

FEM simulations are implemented in MATLAB.