Physics-Informed Neural Networks (PINNs) for Parameterized PDEs: A Metalearning Approach

Michael Penwarden[a], Shandian Zhe[b], Akil Narayan[c], Robert M. Kirby[a]

[a]*School of Computing and Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT*
[b]*School of Computing, University of Utah, Salt Lake City, UT*
[c]*Department of Mathematics and Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT*

Abstract

Physics-informed neural networks (PINNs) as a means of discretizing partial differential equations (PDEs) are garnering much attention in the Computational Science and Engineering (CS&E) world. At least two challenges exist for PINNs at present: an understanding of accuracy and convergence characteristics with respect to tunable parameters and identification of optimization strategies that make PINNs as efficient as other computational science tools. The cost of PINNs training remains a major challenge of Physics-informed Machine Learning (PiML) – and, in fact, machine learning (ML) in general. This paper is meant to move towards addressing the latter through the study of PINNs for parameterized PDEs. Following the ML world, we introduce *metalearning* of PINNs for parameterized PDEs. By introducing metalearning and transfer learning concepts, we can greatly accelerate the PINNs optimization process. We present a survey of *model-agnostic* metalearning, and then discuss our *model-aware* metalearning applied to PINNs. We provide theoretically motivated and empirically backed assumptions that make our metalearning approach possible. We then test our approach on various canonical forward parameterized PDEs that have been presented in the emerging PINNs literature.

Keywords: Physics-Informed Neural Networks (PINNs), metalearning

Email addresses: mpenwarden@sci.utah.edu. (Michael Penwarden), zhe@cs.utah.edu (Shandian Zhe), akil@sci.utah.edu (Akil Narayan), kirby@cs.utah.edu (Robert M. Kirby)

1. Introduction

Physics-informed Machine Learning (PiML) represents the modern confluence of two powerful computational modeling paradigms: data-intensive machine learning concepts and model-informed simulation science. Physics-informed Neural Networks (PINNs) [1, 2, 3] as a means of discretizing partial differential equations (PDEs) are garnering much attention in the Computational Science and Engineering (CS&E) world. PINNs represent a new "meshfree" discretization methodology built upon deep neural networks (DNNs) and capitalizing on machine learning technologies such as automatic backward differentiation and stochastic optimization [4]. The marriage of computational modeling and machine learning is predicted to transform the way we do science, engineering, and clinical practice [5]. Some argue that they augment existing items in our computational science toolbox, while others claim they may supplant traditional methods such as finite elements, finite volumes, and finite differences. Regardless, at least two challenges exist for PINNs: an understanding in a tunable way of their accuracy and convergence characteristics and optimizations that make PINNs as efficient as other computational science tools.

We focus on the latter issue: in their current state, PINNs often are far more expensive to train than the computational (often linear algebra-based) solver kernels of other traditional methods. PINNs proponents argue that traditional methods, particularly the area of computational linear algebra, have had over forty years of concerted effort expended towards their development and optimization and that the comparison will only be fair when the same amount of energy is used expended towards accelerating PiML. Correspondingly, they argue, it is not surprising that in the nascent field of PiML, the cost of PINNs training remains a major challenge of physics-informed machine learning (PiML) – and in fact, machine learning (ML) in general – during these early days of development.

This paper is meant to move towards addressing the optimization challenge previously described through the study of PINNs for parameterized PDEs. Following the ML world, we introduce *metalearning* of PINNs for parameterized PDEs. By introducing metalearning and transfer learning concepts, we can greatly accelerate the PINNs

optimization process. We present a survey of *model-agnostic* metalearning and transfer learning concepts and then discuss our *model-aware* specialization of metalearning applied to PINNs. We provide theoretically motivated and empirically backed assumptions that make our metalearning approach possible. We then test our approach on various canonical forward parameterized PDEs that have been presented in the emerging PINNs literature.

The paper is organized as follows: In Section 2, we provide an overview of metalearning and transfer learning and then explain how finding solutions over parameterized PDEs can be viewed as a metalearning problem. In Section 3, we first review the original PINNs collocation approach and provide a summary of current and ongoing PINNs efforts within the field. Although we focus on the application of our metalearning approach to the collocation version of PINNs, nothing precludes the extension of our work to other PINNs variants with appropriate generalization and minor modifications. In Section 4, we survey two common categories of methods applied to parameterized PDE reduced-order modeling – statistical methods for regression and numerical methods for building approximations, and show how both categories can be viewed in the PINNs context as *model-aware* metalearning algorithms. In Section 5, we present our metalearning PINNs approach applied to forward parameterized PDE problems that have been presented in the emerging PINNs literature. Furthermore, we discuss our approach's limitations and assumptions with open theoretical and methodological challenges to the PINNs and machine learning communities. We conclude in Section 6 with a summary of our work and a discussion of current challenges and potential future avenues of inquiry and expansion of the concepts presented in this work.

2. Parameterized PDEs and Metalearning

In this section, we first present an overview, from the machine learning perspective, of the emerging areas of metalearning and transfer learning. Subsequently in this paper we use the term *metalearning* to refer to both (although we acknowledge that there are nuanced and important differences). We then discuss how computing solutions of parameterized PDEs can be viewed as a metalearning problem.

2.1. Overview of Metalearning and Transfer Learning

Metalearning [6, 7] is a class of machine learning methodologies that intends to quickly adapt a learning model to new tasks or environments, mimicking human learning. To this end, metalearning usually extracts some common, important meta information, such as the initial values of the model parameters and other hyperparameters from a family of training tasks that are correlated to new, unseen tasks. This meta information is used to conduct the training on the new tasks, which is expected to accelerate the training and/or improve the performance. Metalearning is a cross-disciplinary research area between multitask learning [8] and transfer learning [9, 10]. Current metalearning approaches can be (roughly) classified into three categories: (1) metric-learning methods that learn a metric space (in the outer level), where the tasks (in the inner level) make predictions by simply matching the training points, e.g., nonparametric nearest neighbors [11, 12, 13, 14, 15]; (2) black-box methods that train feed-forward or recurrent neural networks (RNNs) to take the hyperparameters and task dataset as the input and predict the optimal model parameters or parameter updating rules [16, 17, 18, 19, 20, 21, 22, 23, 24]; and (3) optimization-based methods that conduct a bi-level optimization, where the inner level is to estimate the model parameters given the hyperparameters and the outer level is to optimize the hyperparameters via a meta-loss [25, 26, 27, 28, 29, 30, 31, 32]. Other (hybrid) approaches include [33, 34], etc. An excellent survey about metalearning for neural networks is given in [35].

2.2. Model-Agnostic Metalearning

We discuss in this section a popular and effective metalearning strategy. Suppose we are interested in a family of correlated learning tasks $\mathcal{A}$, over which we will employ a machine learning model $\mathcal{M}$, e.g., a deep neural network. Note that $\mathcal{A}$ can be an infinite set. A particularly successful metalearning algorithm is model-agnostic metalearning (MAML) [25] that aims to find an initialization $\boldsymbol{\theta}$ for training $\mathcal{M}$ that can well adapt to all the tasks in $\mathcal{A}$. To this end, MAML samples N tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$ from $\mathcal{A}$, and for each $\mathcal{T}_n$, collects a dataset $\mathcal{D}_n$. Each $\mathcal{D}_n$ is further partitioned into a metatraining dataset $\mathcal{D}_n^{\text{tr}}$ and a meta-validation dataset $\mathcal{D}_n^{\text{val}}$. To evaluate the task learning performance with $\boldsymbol{\theta}$ as the initialization, MAML performs one-step (or a few steps

of) gradient descent with the loss on $\mathcal{D}_n^{\text{tr}}$, and then evaluates the loss on $\mathcal{D}_n^{\text{val}}$,

$$\mathcal{L}(\boldsymbol{\theta} - \eta \nabla \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_n^{\text{tr}}), \mathcal{D}_n^{\text{val}}), \tag{1}$$

where η is the step size. This essentially assumes that the model performance after one-step (or a few steps of) update from the initialization can well describe the performance after more thorough training. Accordingly, MAML minimizes the following meta-objective to optimize $\boldsymbol{\theta}$,

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{\mathcal{T}_n \in \mathcal{S}} \mathcal{L}\big(\boldsymbol{\theta} - \eta \nabla \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_n^{\text{tr}}), \mathcal{D}_n^{\text{val}}\big). \tag{2}$$

We use MAML as the baseline metalearning method against which to compare in the results section.

2.3. Parameterized PDEs as a Metalearning Problem

We now describe how solutions to parameterized PDEs can be viewed as a metalearning problem. In brief, we view different parameter values or regions as a collection of tasks; data for learning parametric behavior is gathered as snapshot (fixed-parameter) solutions to the PDE. Consider a general parameterized nonlinear system of steady-state or transient PDEs of arbitrary order for the dependent scalar variable $u(\boldsymbol{x}, t; \boldsymbol{\xi})$,

$$\begin{cases} \frac{\partial}{\partial t} u(\boldsymbol{x}, t; \boldsymbol{\xi}) + \mathcal{F}\left(u(\boldsymbol{x}, t; \boldsymbol{\xi})\right) = \mathcal{S}(\boldsymbol{x}, t; \boldsymbol{\xi}), & \Omega \times [0, T] \times \mathcal{X}, \\ \mathcal{B}(u; \boldsymbol{\xi}) = 0, & \partial\Omega \times [0, T] \times \mathcal{X}, \\ u(x, 0; \boldsymbol{\xi}) = u_0(x; \boldsymbol{\xi}), & \Omega \times \{t = 0\} \times \mathcal{X}, \end{cases} \tag{3}$$

where $\mathcal{F}$ is a nonlinear operator that may contain parameters $\boldsymbol{\xi} \in \mathcal{X} \subset \mathbb{R}^m$. $\mathcal{S}$ is the source term/function, Ω and T are the spatial and temporal domain of interest, $\mathcal{B}$ is the boundary condition operator also potentially parameterized via $\boldsymbol{\xi}$, and $u_0(\boldsymbol{x}, \boldsymbol{\xi})$ parameterizes the initial condition. The variable $\boldsymbol{x} \in \Omega \subset \mathbb{R}^s$ is the spatial coordinate of an s-dimensional space and $t \in [0, T]$ represents the temporal variable. The PDEs can be fully nonlinear and parameterized in an arbitrary fashion (including the initial and boundary conditions); we assume the parametric solution map is well-posed, i.e.,

the solution map $\boldsymbol{\xi} \mapsto u(\cdot, \cdot; \boldsymbol{\xi})$ is a well-defined map from $\mathcal{X}$ to an appropriate function space over $\Omega \times [0, T]$. Conceptually, based upon the reasoning in [36], we seek to define a manifold of solutions over a parameter space of dimension m.

For a given parameter $\boldsymbol{\xi}$, the system (3) can be numerically solved for $u(\boldsymbol{x}, t)$ typically using an approximate solver, e.g., based upon finite difference, finite element or finite volume methods. For applications such as uncertainty quantification (UQ) (where $\boldsymbol{\xi}$ encodes the randomness) and design optimization (where $\boldsymbol{\xi}$ encodes the design parameters), a large number of such forward evaluations is required. Direct implementation via a numerical solver can be computationally prohibitive, especially in the many-query contexts of design and UQ. In the subsequent section (Section 3), we will highlight the use of PINNs as an alternative numerical solver for these purposes, which will be our computational strategy for sampling from the solution manifold.

A direct approximation of $u(\boldsymbol{x}, t; \boldsymbol{\xi})$ is difficult due to the number of samples we need to cover the response surface for such a high-dimensional input space problem [37]. Following the pioneer work of Higdon et al. [37], we can record values at specified (fixed) spatial locations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_x}$ and temporal locations $t_1, \ldots, t_{N_t}$ to provide a discrete approximation of $u(\boldsymbol{x}, t; \boldsymbol{\xi})$. With the recording coordinates, our quantity of interest becomes a vectorial function of the PDE parameters

$$\boldsymbol{y}(\boldsymbol{\xi}) = (u(\boldsymbol{x}_1, t_1; \boldsymbol{\xi}), \ldots, u(\boldsymbol{x}_{N_x}, t_1; \boldsymbol{\xi}), u(\boldsymbol{x}_1, t_2; \boldsymbol{\xi}), \ldots, u(\boldsymbol{x}_{N_x}, t_{N_t}; \boldsymbol{\xi}))^\top \in \mathbb{R}^d,$$

where $\boldsymbol{x}_i$ is a spatial coordinate of a regular/irregular grid, and $d = N_x \times N_t$ is the total number of spatial-temporal grid points.[1] Note that although we use the language of solution evaluation, all of these definitions can be modified to denote spatial and temporal degrees of freedom more broadly (e.g. Fourier modes, etc.). In the subsequent section (Section 3), these evaluation points will be related to the collocation points of the PINNs solution.

Following [36], the challenge in general becomes how to approximate the mapping $\boldsymbol{y}(\boldsymbol{\xi}) : \mathcal{X} \to \mathbb{R}^d$ with a limited computational budget. There is a rich literature around

[1] We assume that $u(x, t; \boldsymbol{\xi})$ is scalar-valued, but straightforward extensions can be considered for systems of PDEs.

the topic of approximating the solution manifold over the parameters; we refer the reader to [38] which both surveys and summarizes these methods. Furthermore, in recent years, methods from both the statistical and/or machine learning communities have been proposed, e.g. [39, 40, 41].

Using the language of Section 2.1, we now recast the problem statement above into a *metalearning* problem. Assume we start with a sampling of the parameter space $\boldsymbol{\xi}_i \in \mathcal{X}$, $i = 1, \ldots, K$ at which we evaluate $u(\boldsymbol{x}, t; \boldsymbol{\xi})$ (or its discretized version $\boldsymbol{y}(\boldsymbol{\xi})$). The particular choice of sampling will depend on what mathematical or statistical properties may be needed in subsequent computation. Since different parameters denote different specific PDEs, each parameter choice, in the extreme, can be considered a *task*. However, often non-overlapping subsets of the parameter space $\mathcal{X}_i \subset \mathcal{X}$, $i = 1, \ldots, K$, the union of which which amounts to the full parameter space, represent a finite collection of tasks. In this case, tasks denote regions of the parameter space over which the mathematical characteristics of the PDE remain the same. For example, consider the compressible Navier-Stokes equations for which Mach number Ma and Reynolds number Re denote the parameter space. Partitioning values of Ma according to the boundary $Ma = 1$ may clearly denote two different parameter spaces for which the mathematical properties of the PDE differ. Based upon these observations, one can appreciate that although significant research within the machine learning domain has been placed on *model-agnostic* metalearning, using this paradigm we can employ various reduced-order modeling techniques to generate *model-aware* metalearning. In the next section, we will first review summarize PINNs and how it fits into this framework, and then state both the theoretical and implementation considerations that are necessary to successfully employ this view of parameterized PDE metalearning.

3. Physics-Informed Neural Networks (PINNs)

In this section, we first present a review of physics-informed neural networks (PINNs), with an emphasis on the original collocation PINNs approach which we use in this work. We also provide a brief summary of current and ongoing PINNs efforts within

the field – many if not all of which might benefit from the metalearning approach presented herein. We then present the application of PINNs metalearning to parameterized PDEs. We first provide a summary of the theoretical considerations upon which our work is built, and subsequently we provide a summary of the implementation considerations that are required.

3.1. Review of Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs) were originally proposed by Karniadakis and co-workers [1, 2, 3] as a neural network based alternative to traditional PDE discretizations. In the original PINNs work, when presented with a PDE specified over a domain Ω with boundary conditions on $\partial\Omega$ and initial conditions at $t = 0$ (in the case of time-dependent PDEs), the solution is computed (i.e. the differential operator is satisfied) as in other mesh-free methods like RBF-FD [42, 43] at a collection of collocation points. First, we re-write our PDE system in residual form as $R(u) = \mathcal{S} - \frac{\partial}{\partial t}u - \mathcal{F}(u)$, where $\mathcal{S}$ and $\mathcal{F}$ are as defined in (3). The PINNs formulation is expressed as follows: Given a neural network function $\tilde{u}(\boldsymbol{x}, t; \underline{\mathrm{w}})$ with specified activation functions and a weight matrix $\underline{\mathrm{w}}$ denoting the degrees of freedom derived from the width and depth of the network, find $\underline{\mathrm{w}}$ that minimizes the loss function:

$$MSE = MSE_u + MSE_R \tag{4}$$

where

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} \|\tilde{u}(x_u^i, t_u^i; \underline{\mathrm{w}}) - u^i\|^2 \tag{5}$$

$$MSE_R = \frac{1}{N_R}\sum_{i=1}^{N_R} \|R(\tilde{u}(x_R^i, t_R^i)\|^2 \tag{6}$$

where $\{x_u^i, t_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $u(\boldsymbol{x}, t)$ and $\{x_R^i, t_R^i\}_{i=1}^{N_R}$ specify the collocation points for evaluation of the collocating residual term $R(\tilde{u})$. The loss MSE_u corresponds to the initial and boundary data while MSE_R enforces the structure imposed by the differential operator at a finite set of collocation

points. This loss-function modified minimization approach fits naturally into the traditional deep learning framework [4]. Various optimization choices are available including stochastic gradient descent (SGD), L-BFGS, etc. The result of applying this procedure is a neural network $\tilde{u}(\boldsymbol{x}, t; \underline{\mathrm{w}})$ that attempts to minimize through a balancing act the strong imposition of the initial and boundary conditions and satisfaction of the PDE residual. Note that this statement does not immediately connect to the approximation error $\|u(\boldsymbol{x}, t) - \tilde{u}(\boldsymbol{x}, t; \underline{\mathrm{w}})\|$; however, consistency and convergence are items of current research (e.g. [44]).

Beyond the initial collocation version of PINNs expressed above, Karniadakis and collaborators have extended these methods to conservative PINNs (cPINNs) [45], variational PINNs (vPINNS) [46], parareal PINNs (pPINNs) [47], stochastic PINNs (sPINNs) [48], fractional PINNs (fPINNs) [49], LesPINNs (LES PINNs) [50], non-local PINNs (nPINNs) [51] and eXtended PINNs (xPINNs) [52].

In this work, we will focus on application of the original collocation PINNs approach; however, the work presented herein can be applied to many if not all of these variants.

3.2. Application of PINNs Metalearning to Parameterized PDEs

This section provides a connection between metalearning and parameterized PDEs that we will exploit for PINNs algorithms. To begin, we provide a more detailed overview of the anatomy of a PINN that complements the high-level discussion of Section 3.1.

In the context of parameterized PDEs (Section 2.3), we consider an arbitrary but fixed value $\boldsymbol{\xi}$ of the parameter. A PINN emulator is the map $(\boldsymbol{x}, t) \mapsto \tilde{u}(\boldsymbol{x}, t; \underline{\mathrm{w}})$, where we assume that the weights $\underline{\mathrm{w}}$ are trained as discussed in Section 3.1. The function $\tilde{u}$ is a neural network, i.e., an iterative compositions of affine maps and an activation function σ; in the special case of PINNs this can be summarized as

$$
\begin{aligned}
h_0 &= (\boldsymbol{x}, t) \\
h_j &= \sigma\left(W_i h_{j-1} + b_j\right) \qquad\qquad j = 1, \ldots, \ell + 1 \\
\tilde{u} &= h_{\ell+1}
\end{aligned}
$$

where $h_j \in \mathbb{R}^{n_j}$ are hidden states, $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function that is applied componentwise to vectors, $b_j \in \mathbb{R}^{n_j}$ are bias vectors, and $W_j \in \mathbb{R}^{n_j \times n_{j-1}}$ are weight matrices. The network has ℓ hidden layers, and the width of layer j is n_j. The input layer $(\boldsymbol{x}, t)$ is of width $n_0 = p + 1$, and the output layer is of width $n_{\ell+1} = 1$. With $\underline{\mathbf{W}}_j \coloneqq [W_j \;\; b_j] \in \mathbb{R}^{n_j \times (n_{j-1}+1)}$ the concatenation of weights and biases in layer j, then we collect all the parameters of the neural network in the *weight* vector,

$$\underline{\mathrm{w}} \coloneqq \left(\mathrm{vec}(\underline{\mathbf{W}}_1)^\top, \;\; \ldots, \;\; \mathrm{vec}(\underline{\mathbf{W}}_{\ell+1})^\top\right)^\top \in \mathbb{R}^{M_\ell}, \qquad M_\ell = \sum_{j=1}^{\ell+1} n_j(n_{j-1}+1).$$

We have assumed a simple network architecture above, e.g., fully connected and with the same activation function for all neurons. However, the discussion can be appropriately extended to more customized network architectures. In the future, we will refer to $\underline{\mathrm{w}}$ as the weights that are trained through the procedure in Section 3.1.

In the context of metalearning, the map $\boldsymbol{\xi} \mapsto \underline{\mathrm{w}}$ is the task associated to parameter $\boldsymbol{\xi}$. With this in mind, we will write $\underline{\mathrm{w}} = \underline{\mathrm{w}}(\boldsymbol{\xi})$ to explicitly communicate that the trained weights depend on $\boldsymbol{\xi}$. The metalearning problem here is, given a parameter $\boldsymbol{\xi}$, identify a set of weights that is well-adapted to the task of identifying $\underline{\mathrm{w}}(\boldsymbol{\xi})$. More precisely, given K sample task parameters $\{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_K\} \subset \mathbb{R}^m$, compute a task predictor $\underline{\hat{\mathrm{w}}}(\boldsymbol{\xi})$ satisfying $\underline{\hat{\mathrm{w}}}(\boldsymbol{\xi}) \approx \underline{\mathrm{w}}(\boldsymbol{\xi})$ for all parameters $\boldsymbol{\xi} \in \mathcal{X}$. In this paper, we build $\underline{\hat{\mathrm{w}}}$ as a linear map,

$$\underline{\hat{\mathrm{w}}}(\boldsymbol{\xi}) = \sum_{k=1}^{K} \boldsymbol{\psi}_k c_k(\boldsymbol{\xi}), \tag{7}$$

where $\boldsymbol{\psi}_k$ are vectors in $\mathbb{R}^{M_\ell}$, and $c_k(\cdot)$ are coefficient functions. We will utilize various approaches that will specify the vectors $\boldsymbol{\psi}_k$ and coefficients c_k via available sample task outputs $\{\underline{\mathrm{w}}(\boldsymbol{\xi}_1), \ldots, \underline{\mathrm{w}}(\boldsymbol{\xi}_N)\}$. Our mathematical approaches will treat c_k as deterministic functions, whereas statistical approaches treat c_k as random variables.

3.2.1. Theoretical Considerations

We provide discussion in this section that motivates why a metalearning ansatz of the form (7) can be successful for our core problem of approximating solutions to parametric PDEs with PINNs. Our core assumption is that learning the $\boldsymbol{\xi}$-variation of

weights in a PINN is analogous to learning such parametric variation of the solution u itself. We cannot yet make this analogy quantitatively concrete since the behavior of parameters of a trained neural network is still an area of active research. Nevertheless, under this assumption, we analogize the task of learning $\boldsymbol{\xi}$ variation of neural network weights to that of learning $\boldsymbol{\xi}$ variation of the PDE solution u, and the latter problem is much more well-understood.

We first note that for several parametric PDEs of interest, the solution map $\boldsymbol{\xi} \mapsto u(\cdot, \cdot; \boldsymbol{\xi})$ is (real and/or complex) analytic. This fact is known to result in encouraging theoretical guarantees for approximations of parametric PDEs [53]. To be more precise, we furnish two examples.

First, under the previously described analyticity assumptions, and assuming $\mathcal{X} = [-1, 1]^m$, one can conclude the existence of an approximation u_N satisfying,

$$u_K \coloneqq \sum_{n=1}^{K} v_n \phi_n(\boldsymbol{\xi}), \qquad \|u(\boldsymbol{\xi}) - u_K(\boldsymbol{\xi})\|_{F(\Omega \times [0,T])} \lesssim \exp\left(-K^{1/m}\right) \tag{8}$$

uniformly for $\boldsymbol{\xi} \in \mathcal{X}$. Above, $F(\Omega \times [0, T])$ is an appropriate function space over the spacetime variables, the v_n are function-valued coefficients, and ϕ_n are *polynomial* functions of $\boldsymbol{\xi}$. Hence, polynomial approximations can, in principle, yield exponential convergence with efficacy blunted by the curse of dimensionality, as evidenced by the $K^{1/m}$ exponent. We emphasize the parallel between u_K in the expression above and $\underline{\hat{\mathrm{w}}}$ in (7).

As a second example, one can gain even better rates of convergence, at the cost of making more restrictive technical assumptions. In this case, a similar type of approximation u_N yields *dimension-independent* convergence rates,

$$\|u(\boldsymbol{\xi}) - u_K(\boldsymbol{\xi})\|_{F(\Omega \times [0,T])} \lesssim K^{-r}, \qquad r = \frac{1}{q} - 1, \;\; 0 < q < 1,$$

again, uniformly for $\boldsymbol{\xi} \in [-1, 1]^m$, where q is a constant that depends on the PDE, but not on x, t, or $\boldsymbol{\xi}$. Above, the particular u_N approximation differs from the one achieving (8), but the form of the approximation is identical and in particular utilizes basis functions ϕ_n that are still polynomials. The estimate above applies to a particular class of parametric elliptic PDEs under certain assumptions which determine the constant q. We refer the reader to [54, Corollary 7.4] for these assumptions. The relevant message

in our context is that q is *independent* of the parameter dimension m, demonstrating that it is possible, in principle, to efficiently approximate solutions to parametric PDEs with linear maps of a form similar to (7).

In the PINNs context, our emulator (7) does not approximate the solution map itself so that the above theory does not apply. Instead, we build an emulator on the parameters of a neural network. While the behavior of $\boldsymbol{\xi}$-variability of trained PINNs parameters is not yet fully understood, the theory above suggests that in some situations, one can expect that linear approximations such as (7) can perform well in approximating the manifold of well-trained neural network parameters, under the assumption that the trained weights exhibit a similar variation. Our numerical results section explores cases when this indeed is successful, either with polynomials or with other types of approximation procedures.

3.2.2. Implementation Considerations

The PINN metalearning approach laid out in Section 3.2 contains three steps, summarized as: (1) sample the parameter space appropriately for the determined weight prediction method; (2) evaluate the PINN at the sampled locations and store trained weights; (3) construct the weight prediction model. When queried at a new location, this predictive method should output a reasonable estimation of what the trained PINN weights would be, thus reducing the optimization time. Based on this, we must decide how to sample the space and what prediction method to use. In this paper, we present five methods with appropriate sampling for each, implying that we are not advocating any particular method over another but rather to show there are many ways this can be approached. We also compare this to the standard randomization of weights, MAML, and a zeroth-order model in which we initialize with the trained weights at the center of the parameter space $\mathcal{X}$. Any method or sampling could be chosen to replace these steps.

Additionally, it is essential to ensure the weights vary smoothly across $\mathcal{X}$ as the methods we are using assume smoothness. We have found that initializing the training sets with the trained weights at the center of the parameter space appears to empirically ensure smoothness for one type of PDE task as mentioned in Section 2.3. We

are attempting to start the weights optimization in the same basin of attraction for all samples by doing this. We assume that by doing this, we obtain smooth weights over $\mathcal{X}$, but this is not guaranteed. Alternatively, initializing with random weights ensures one will fall into many different regions of attraction, and the weights will likely not vary smoothly, and so metalearning with our strategy could suffer in accuracy. Future work will be to develop a more sophisticated way to ensure this condition as well as to discover task boundaries so that the method can be task-agnostic.

3.2.3. Algorithmic Complexity

To aid in evaluating the algorithmic complexity and reproducibility of our approach, we provide Algorithm 1. The primary consideration when evaluating the cost of our approach is the number of training points K as we must train PINNs to obtain the weight data for each point. In this paper, we set K across for all methods as the size of the hyperbolic cross set, which is dependent on the parameter dimension m of $\mathcal{X} \subset \mathbb{R}^m$ and the order of the hyperbolic cross set d. These points are used in conjunction with the polynomial least-squares method, and for the other methods, we use a Latin hypercube sampling of size K. We use the same K value for both sampling methods to make a fair comparison, but one can choose any K. Finding a balance of cost associated with solving K PINNs to construct the models with and the cost savings of the models during testing is the main decision complexity in our approach.

4. Methods

In this section, we present two categories of methods within the CS&E world often applied to parameterized PDE reduced-order modeling: statistical methods for regression and numerical methods for approximation. We only select exemplars from both lists for our study; we acknowledge that many more methods exist. However, we hold that these five methods help give sufficient breadth of application and insight to motivate PINNs metalearning.

Algorithm 1 Metalearning PINN Weights Method

Given appropriate PDE set-up information (i.e. spatial domain Ω, temporal domain T , parameter space $\mathcal{X}$, etc.)

Define PINN hyper-parameters (size and optimization) for $\tilde{u}$

Set $\boldsymbol{\xi}$ as the appropriate sampling of $\mathcal{X}$ with size K for the weight prediction model used

Let $\boldsymbol{\xi}^{\mathcal{C}}$ be the centroid of $\mathcal{X}$

Store trained weights $\underline{\mathrm{w}}(\boldsymbol{\xi}^{\mathcal{C}})$ from $\tilde{u}(\boldsymbol{x}, t; \boldsymbol{\xi}^{\mathcal{C}})$

Assume trained PINN weights $\underline{\mathrm{w}}(\boldsymbol{\xi})$ are smooth across $\mathcal{X}$ if initialized with $\underline{\mathrm{w}}(\boldsymbol{\xi}^{\mathcal{C}})$

for j = 1 **to** K **do**

  Initialize $\tilde{u}$ weights $\underline{\mathrm{w}}(\boldsymbol{\xi}_j)$ with $\underline{\mathrm{w}}(\boldsymbol{\xi}^{\mathcal{C}})$

  Store vectorized trained weights $\underline{\mathrm{w}}(\boldsymbol{\xi}_j)$ from $\tilde{u}(\boldsymbol{x}, t; \boldsymbol{\xi}_j)$

end for

Construct weight prediction models $\underline{\hat{\mathrm{w}}}(\boldsymbol{\xi})$ (GPs, RBF, etc.) from stored vectorized weight data $\underline{\mathrm{w}}(\boldsymbol{\xi})$

Predict PINN weights for any point $\underline{\hat{\mathrm{w}}}(\boldsymbol{\xi}^*)$ to initialize a PINN with, then train

4.1. Statistical Methods for Regression

Gaussian processes (GPs) are powerful statistic regression models. Due to their nonparametric Bayesian nature [55], GPs can automatically capture the complexity of the functions underlying the data, and quantify the predictive uncertainty via a closed form (i.e., Gaussian). The standard GP learns a single-output stochastic process $f : \mathbb{R}^m \to \mathbb{R}$ from the training data $\mathcal{D} = \{(\boldsymbol{\xi}_1, y_1), \ldots, (\boldsymbol{\xi}_K, y_K)\}$ where each $\boldsymbol{\xi}_k$ is an input vector and $\boldsymbol{\Xi} = (\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_K)$. The function values $\mathbf{f} = (f(\boldsymbol{\xi}_1), \ldots, f(\boldsymbol{\xi}_K))$ are assumed to follow a multivariate Gaussian distribution — the finite projection of a Gaussian process $\mathcal{N}$ on the training inputs — $p(\mathbf{f}|\boldsymbol{\Xi}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ are the mean function values of every input and usually set to $\mathbf{0}$, and the covariance matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} \kappa(\boldsymbol{\xi}_1, \boldsymbol{\xi}_1) & \cdots & \kappa(\boldsymbol{\xi}_1, \boldsymbol{\xi}_K) \\ \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{\xi}_K, \boldsymbol{\xi}_1) & \cdots & \kappa(\boldsymbol{\xi}_K, \boldsymbol{\xi}_K) \end{pmatrix}$$

where κ is a kernel function of the input vectors. The observed outputs $\boldsymbol{y}$ are assumed to be sampled from a noisy model, e.g., $p(\boldsymbol{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{y}|\mathbf{f}, \tau\mathbf{I})$. Integrating out $\mathbf{f}$, we obtain the marginal likelihood $p(\boldsymbol{y}|\boldsymbol{\Xi}) = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \boldsymbol{\Sigma} + \tau\mathbf{I})$. Computational estimation of kernel parameters and the noise variance τ is frequently achieved via maximum likelihood estimation.

To predict the output for a test input $\boldsymbol{\xi}^*$, we use the conditional Gaussian distribution, since the test and training outputs jointly follow a multivariate Gaussian distribution.

$$\begin{aligned} \text{We have } & p\big(f(\boldsymbol{\xi}^*)|\boldsymbol{\xi}^*, \boldsymbol{\Xi}, \boldsymbol{y}\big) = \mathcal{N}\big(f^*|\alpha(\boldsymbol{\xi}^*), \beta(\boldsymbol{\xi}^*)\big) \\ \text{where } & \alpha(\boldsymbol{\xi}^*) = \kappa_*^\top (\boldsymbol{\Sigma} + \tau^{-1}\mathbf{I})^{-1}\boldsymbol{y}, \\ & \beta(\boldsymbol{x}^*) = \kappa(\boldsymbol{\xi}^*, \boldsymbol{\xi}^*) - \kappa_*^\top (\boldsymbol{\Sigma} + \tau^{-1}\mathbf{I})^{-1}\kappa_*, \\ & \kappa_* = [\kappa(\boldsymbol{\xi}^*, \boldsymbol{\xi}_1), \ldots, \kappa(\boldsymbol{\xi}^*, \boldsymbol{\xi}_K)]^\top. \end{aligned}$$

In particular, we use the mean of $p\big(f(\boldsymbol{\xi}^*)|\boldsymbol{\xi}^*, \boldsymbol{\Xi}, \boldsymbol{y}\big)$ as the predictor at $\boldsymbol{\xi}^*$. These methods are implemented with GPyTorch [56].

4.1.1. Multi-task Gaussian Process Modeling

Many tasks require learning a function with multiple output.[2] A classical multi-output regression framework is multi-task GP [57]. It models the outputs of all the tasks as a single GP, where the kernel function between arbitrary two outputs is defined as

$$g([\boldsymbol{\xi}_1, t_1], [\boldsymbol{\xi}_2, t_2]) = \kappa(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) \cdot s_{t_1 t_2}, \tag{9}$$

where $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2$ are the inputs and t_1, t_2 are the task indices, $\kappa(\cdot, \cdot)$ is a kernel function of the inputs, and $s_{t_1 t_2}$ is the task similarity between t_1 and t_2. The training is the same as the standard GP. However, we need to estimate not only the parameters of $\kappa(\cdot, \cdot)$, but also the similarity values between each pair of tasks $\{s_{t_1 t_2}\}$.

4.1.2. Linear Method of Coregularization (LMC)

Another successful multi-output regression model is the Linear Model of Coregionalization (LMC) [58], which assumes the outputs are a linear combination of a set of basis vectors weighted by independent random functions. We introduce Q bases $\boldsymbol{\Psi} = [\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_Q]^\top$ and model a M_ℓ-dimensional vector function by

$$\underline{\hat{w}}(\boldsymbol{\xi}) = \sum_{q=1}^{Q} \boldsymbol{\psi}_q c_q(\boldsymbol{\xi}) = \boldsymbol{\Psi}^\top \mathbf{c}(\boldsymbol{\xi}) \tag{10}$$

where Q is often chosen to be much smaller than M_ℓ, and the random weight functions $\mathbf{c}(\boldsymbol{\xi}) = [c_1(\boldsymbol{\xi}), \ldots, c_Q(\boldsymbol{\xi})]^\top$ are sampled from independent GPs. In spite of a linear structure, the outputs $\underline{\hat{w}}$ are still nonlinear to the input $\boldsymbol{\xi}$ due to the nonlinearity of the weight functions. LMC can easily scale up to a large number of outputs: once the bases $\boldsymbol{\Psi}$ are identified, we only need to estimate a small number of GP models ($Q \ll M_\ell$). For example, we can perform PCA on the training outputs to find the bases, and use the singular values as the outputs to train the weight functions. This is also referred to as PCA-GP [37].

[2] Task here is used differently than in the metalearning section. We use the nomenclature of the GP field here.

4.2. Numerical Approximation Methods for Regression

We describe here common strategies in the numerical approximation community for building emulators of the form (7) from data. We choose to focus on three different types of approaches: First we build approximations via cubic spline interpolants when $\boldsymbol{\xi}$ is one or two dimensional. The second class of methods use "meshless" radial basis function (RBF) methods with various kernels. The third class of methods are polynomial approximation schemes using least squares approximation.

We briefly describe these approaches in each section below, focusing on specifying how the coefficients $\boldsymbol{\psi}_k$ and basis functions $c_k(\boldsymbol{\xi})$ in (7) are chosen and computed. In all the methods, we assume availability of data $\underline{\mathrm{w}}(\boldsymbol{\xi}_j)$ (i.e., trained PINNs network parameters) on a discrete sampling $\{\boldsymbol{\xi}_i\}_{i=1}^K$. We will specify how this grid is chosen in each section.

4.2.1. Cubic Spline Interpolation

Univariate cubic splines are smooth interpolants of data [59]. More precisely, with $(\boldsymbol{\xi}_i)_{i=1}^K$ an ordered set of points from a Latin Hypercube Sampling (LHS) design [60] on the one-dimensional interval $\mathcal{X}$, then a cubic spline approach builds the approximation $\underline{\hat{\mathrm{w}}}$ in (7) as a piecewise cubic function, where $\underline{\hat{\mathrm{w}}}$ is a cubic polynomial on each interval $[\boldsymbol{\xi}_j, \boldsymbol{\xi}_{j+1}]$, and is continuously differentiable at each $\boldsymbol{\xi}_j$. One can build such an approximation in terms of divided differences of the data $\underline{\mathrm{w}}(\boldsymbol{\xi}_j)$, so that the coefficients $\boldsymbol{\psi}_k$ are linear in this data. The basis functions $c_k(\boldsymbol{\xi})$ are then piecewise cubic polynomials. This is implemented with the SciPy package functions scipy.interpolate.UniveriateSpline and scipy.interpolate.SmoothBivariateSpline [61]. The bivariate case does not require a tensoral grid therefore we can use it with our sampling methods with the implemtation caveat that $K \geq (x_{deg} + 1) * (y_{deg} + 1)$ which in the cubic spline case is $K \geq 16$. Additionally, we use the default function parameters.

4.2.2. Radial Basis Function (RBF) Interpolation

RBF approximations build the emulator $\underline{\hat{\mathrm{w}}}$ in (7) using shifted kernels as basis functions,

$$c_k(\boldsymbol{\xi}) = \kappa(\boldsymbol{\xi}, \boldsymbol{\xi}_k),$$

and the coefficients ψ_k are chosen by enforcing interpolation of $\underline{\hat{\mathrm{w}}}$ on the data points [62]. We again choose the parametric grid as an LHS design, and employ three different types of commonly used kernels:

$$\kappa(\boldsymbol{\xi}, \boldsymbol{\zeta}) = r^3, \qquad \text{(Cubic)}$$

$$\kappa(\boldsymbol{\xi}, \boldsymbol{\zeta}) = \exp\left(-\left(\frac{r}{\tau}\right)^2\right), \qquad \text{(Gaussian)}$$

$$\kappa(\boldsymbol{\xi}, \boldsymbol{\zeta}) = \sqrt{\left(\frac{r}{\tau}\right)^2 + 1}, \qquad \text{(Multiquadric)}$$

where $r = \|\boldsymbol{\xi} - \boldsymbol{\zeta}\|_2$ is the Euclidean distance between the inputs $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$, and τ is a tunable parameter, which we choose as the average pairwise distance between grid points.

We use interpolative RBF approximations, which enforce (7) at every data point. Such RBF approximations are closely related to GP statistical models: an RBF approximation equals the mean of a GP approximation built from the same kernel, with zero nugget at the data points.

4.2.3. Polynomial least-squares using hyperbolic cross sets

Our final technique in this section for forming the approximation (7) builds a polynomial approximation via least squares. In this case, the basis functions are polynomials,

$$c_k(\boldsymbol{\xi}) = \boldsymbol{\xi}^{\alpha(k)} = \prod_{j=1}^{m} \left(\xi^{(j)}\right)^{\alpha(k)_j}, \qquad \boldsymbol{\xi} = \left(\xi^{(1)}, \ldots, \xi^{(m)}\right),$$

where $\alpha(k) = (\alpha(k)_1, \ldots, \alpha(k)_m) \in \mathcal{N}_0^m$ is an m-dimensional multi-index on the non-negative integer lattice, and $(\alpha(k))_{k=1}^K$ is any ordering of the multi-indices lying in an order-d *hyperbolic cross* set [63],

$$(\alpha(k))_{k=1}^K =: A = \left\{ \alpha \in \mathcal{N}_0^m \;\middle|\; \prod_{j=1}^{m} (\alpha_j + 1) \leq d + 1 \right\}.$$

Increasing the order d increases the number of terms K in the expansion, and hyperbolic cross sets tend to de-emphasize mixed polynomial terms and therefore have much smaller size than alternative total degree sets. Despite this decreased model capacity,

polynomial approximations on hyperbolic cross sets are known to produce efficient approximations in the context of parametric PDEs [64, 65].

We construct the coefficients ψ_k in (7) using a weighted least squares approach on the data, and the grid is chosen as a set of weighted approximate Fekete points [66]. This method is implemented with the UncertainSCI package [67].

5. Results and Discussion

In this section, we demonstrate the efficacy of our metalearning approach on various representative forward PDE problems. The examples below are extensions of the test problems proposed in [1, 68]. There are two types of spatiotemporal domains we consider: (i) the Burgers' and (nonlinear) Heat equations in which we have a spatial and temporal dimension, and the (ii) Allen-Cahn, Diffusion-Reaction, and linear parametric heat equations in which we have two spatial dimensions.

5.1. Experimental settings

For PDE type (i), the PINN solution test points are a uniform grid of size 256×100 in space and time respectively. For type (ii), we use a uniform grid of size 128×128. These test points are where we compare between the exact solution and the PINN solution. To train the PINN we use 100 uniformly sampled boundary/initial value points for the MSE_u portion of the loss, and $10,000$ collocation points using Latin hypercube sampling (LHS) for the MSE_R portion as described in Section 3.1. In all PINN architectures, we employ fully-connected feed-forward neural networks with tanh activation functions. The width and depth parameters will be specified per experiment below. The values in each table are presented as the empirical mean and standard deviation over the test set.

For all tables, the columns are relative error after 500 iterations of Adam optimization, relative error after L-BFGS optimization, and the time taken to optimize L-BFGS for a certain tolerance condition. The sequential optimization with Adam followed by L-BFGS is standard approach in PINNs. [44]. The L-BFGS conditions are 10^{-6} termination tolerance on first order optimality and 10^{-9} termination tolerance on weight vector changes with a learning rate of 0.1.

For MAML, we train with 1,000 epochs and Adam with learning rate 10^{-3}. The inner optimization uses one-step gradient decent and step-size 10^{-3}. The data used has the same number of boundary points, collocation points, and tasks for the training and validations sets as the other predictive methods presented.

Lastly, we point out that occasionally we train "bad" PINNs where they do not converge relative to their loss function. This has been observed only for random and RBF Gaussian kernel initialization. The random initializations are a byproduct of standard PINNs, and the unconverged runs can be identified with a large loss and re-run. For our predictive RBF Gaussian kernel method, re-running does not help as it provides the same initialization for each point so these were identified via loss and removed. We further discuss RBF Gaussian kernel in the contexts of the results as being a poor method for our metalearning approach. Since the GP approach is a generalization of the RBF approach (it optimizes hyperparameters), then our results suggest the predictable result that optimizing these hyperparameters, which is common in GP methods, appears better than empirically choosing them (e.g., based on fill distances), which is a common workflow in RBF methods.

5.2. 1D Burgers Equation

We consider the following 1D viscous Burgers equation:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial}{\partial x}\left(u^2\right) = \nu \frac{\partial^2 u}{\partial x^2},\ x \in \Omega,\ t \in [0, T] \tag{11}$$

where $\Omega \times [0, T] = [-1, 1], \times [0, 1]$ with the viscosity $\nu \in [0.005, 0.05]$ and initial condition $u(x, 0) = -sin(x\pi)$. For evaluation of the error, we compute the exact solution derived using Cole's transformation computed with Hermite integration [69]. We initially randomly sample our parameter space with 16 PINNs approximations containing 5 hidden layers of width 5. The size 16 derives from the amount of points with the degree 5 hyperbolic cross as described in Section 4.2.3.

Table 1: We sample the PDE parameter space uniformly with 32 points to compute the values in the table.

Initialization Methods	Error after Adam (10^{-3})	Error after L-BFGS (10^{-3})	L-BFGS Time (s)
Random	618.1 ± 177.7	5.1 ± 6.0	146 ± 68
MAML	107.7 ± 40.5	7.4 ± 5.7	184 ± 66
Center	58.6 ± 103.7	6.4 ± 9.1	63 ± 51
Multitask	58.5 ± 84.1	5.2 ± 8.5	53 ± 34
LMC	292.6 ± 98.7	5.0 ± 6.7	58 ± 36
Spline	40.9 ± 65.5	5.9 ± 9.1	40 ± 28
RBF (cubic)	26.1 ± 57.6	5.4 ± 9.3	27 ± 40
RBF (gaussian)	54.2 ± 93.4	5.1 ± 8.2	39 ± 25
RBF (multiquadric)	28.9 ± 63.3	4.6 ± 6.6	26 ± 29
Polynomial	23.5 ± 41.4	5.1 ± 10.1	39 ± 37

In Table 1 we first observe that MAML improves the error after Adam optimization compared to randomized weights; however it worsens the convergence with L-BFGS. Note that Adam and SGD are first-order methods whereas L-BFGS is a second-order method. In PINNs, we further optimize with L-BFGS after Adam which evidently is an issue when using MAML. This may be due to MAML using first-order gradient descent in the inner training loop (see Equation 1), which is inconsistent with switching to second-order methods in practice. MAML appears to do worse than randomized weights in the context of this sequential training in regard to L-BFGS time. In the context of training without L-BFGS, we note that our metalearning method still outperforms MAML after only performing the Adam iterations. In our interpolation methods, we can also see that they greatly improve the time it takes to reach the L-BFGS tolerance, by around a factor of five, while achieving the same accuracy. One trend we note throughout all problems presented is how well initializing with the a run at the center of the parametric domain does for its minimal cost, this will be further discussed.

Remark: There will be some performance variance in these methods depending on the randomization of the weights for the center run. This run is used to initialize the data for interpolation, so it follows that it affects the final outcome. We recommend running the parametric center PINN a few times and taking the most accurate of the learned weights to address this issue. This is also an issue for MAML as it also is dependent on the random weights at which it starts before meta training.

Next, we use a larger PINN containing 8 hidden layers of width 10 to see how this

affects the methods.

Table 2: We sample the PDE parameter space uniformly with 32 points to compute the values in the table.

Initialization Methods	Error after Initializing (10^{-3})	Error after Adam (10^{-3})	Error after L-BFGS (10^{-3})	L-BFGS Time (s)
Random	1245.5 ± 287.3	174.9 ± 130.4	1.2 ± 1.8	105 ± 29
MAML	334.6 ± 37.8	92.3 ± 30.3	1.7 ± 3.2	125 ± 38
Center	70.6 ± 41.6	14.9 ± 38.1	0.9 ± 0.5	56 ± 48
Multitask	96.8 ± 80.9	2.0 ± 2.3	0.7 ± 0.2	24 ± 21
LMC	246.7 ± 32.4	7.8 ± 17.5	0.8 ± 0.4	40 ± 27
Spline	22.7 ± 8.6	1.4 ± 1.7	0.7 ± 0.2	10 ± 11
RBF (cubic)	4.8 ± 4.4	1.5 ± 1.9	0.8 ± 0.4	7 ± 13
RBF (gaussian)	20.0 ± 19.7	10.6 ± 18.2	0.8 ± 0.5	36 ± 30
RBF (multiquadric)	6.1 ± 7.0	1.2 ± 1.2	0.8 ± 0.4	7 ± 8
Polynomial	21.4 ± 10.6	1.2 ± 1.9	0.7 ± 0.3	7 ± 16

In Table 2 we observe that increasing the neural network size not only improves the accuracy but also decreases the time. The analysis of the expressibility of a network is not in the scope of this paper; however, we report this experiment to show our method works regardless of the selection of PINN size. Additionally, we can see that with this setting, most of the methods achieve 10^{-3} error with only Adam optimization, meaning one need not employ the more costly refinement of L-BFGS if the initialization is good. In fact, we also include the column for error after initializing before any optimization is performed. Here we can see the best possible case; we have achieved 10^{-3} error without any training whatsoever for RBF cubic and multiquadratic kernels meaning the network is initialized very well for the PDE parameters.

We observe that, among the RBF approaches, the Gaussian kernel is empirically the worst metalearning strategy. Such poor performance could manifest if the scale parameter of the Gaussian kernel is chosen poorly. In our experiments, we use values that are deterministically computed based on fill distances of the parametric grid, which is known empirically to be a reasonable choice [62]. However, the consistently poor performance of Gaussian RBFs in all the remaining examples we have tested suggests that the Gaussian kernel is simply a poor choice of kernel for parameterized PDE metalearning problems.

5.3. 1D nonlinear Heat Equation

We consider the following 1D nonlinear PDE:

$$\frac{\partial u}{\partial t} - \lambda \frac{\partial^2 u}{\partial x^2} + k\, tanh(u) = f,\ x \in \Omega,\ t \in [0, T] \tag{12}$$

where $\Omega \times [0, T] = [-1, 1] \times [0, 1]$ and where $\lambda \in [1, \pi]$ and $k \in [1, \pi]$ are positive constants. We specify an exact solution of $u(x, t; \lambda, k) = k\, sin(\pi x)\, exp(-\lambda k x^2)\, exp(-\lambda t^2)$ and derive the corresponding form of the forcing f. We initially randomly sample our parameter space with 22 PINNs approximations containing 5 hidden layers of width 10.

Table 3: We sample the PDE parameter space uniformly with 64 points to compute the values in the table.

Initialization Methods	Error after Adam (10^{-3})	Error after L-BFGS (10^{-3})	L-BFGS Time (s)
Random	465.1 ± 218.2	5.5 ± 3.7	178 ± 37
MAML	384.5 ± 167.3	6.2 ± 6.0	211 ± 58
Center	38.4 ± 28.8	4.6 ± 2.8	102 ± 40
Multitask	16.7 ± 14.0	4.6 ± 3.2	60 ± 38
LMC	20.9 ± 16.6	4.6 ± 2.9	66 ± 43
Spline	10.8 ± 18.2	4.7 ± 3.5	43 ± 43
RBF (cubic)	28.5 ± 82.7	4.8 ± 2.8	42 ± 38
RBF (gaussian)	219.0 ± 381.1	4.8 ± 2.8	115 ± 71
RBF (multiquadric)	11.5 ± 18.7	4.3 ± 2.6	33 ± 24
Polynomial	7.4 ± 7.1	4.6 ± 2.7	30 ± 24

In Table 3 we see considerable improvement in optimization with our initialization procedure. We again observe 10^{-3} errors without any need for L-BFGS, in this case with the polynomial interpolation method.

We now consider the same problem but with a degree 2 hyperbolic cross, equating to 13 PINN approximations instead of 22 for degree 5. Because of the conditions for the smooth bivariate spline, a second-order spline is used instead of cubic as described in Section 4.2.1.

Table 4: We sample the PDE parameter space uniformly with 64 points to compute the values in the table.

Initialization Methods	Error after Adam (10^{-3})	Error after L-BFGS (10^{-3})	L-BFGS Time (s)
Random	529.5 ± 465.2	5.2 ± 3.9	156 ± 42
MAML	455.0 ± 180.0	4.5 ± 3.1	188 ± 43
Center	38.8 ± 30.7	4.5 ± 2.4	96 ± 41
Multitask	18.0 ± 18.1	4.8 ± 3.6	49 ± 30
LMC	26.8 ± 28.3	4.9 ± 3.3	59 ± 28
Spline	7.7 ± 6.2	4.7 ± 2.4	29 ± 30
RBF (cubic)	71.2 ± 143.2	4.3 ± 2.0	64 ± 58
RBF (gaussian)	216.5 ± 320.5	5.4 ± 4.0	104 ± 72
RBF (multiquadric)	13.9 ± 19.5	4.4 ± 2.2	35 ± 30
Polynomial	7.3 ± 4.7	4.6 ± 3.2	38 ± 30

In Table 4 we observe that lowering the number of data points used for interpolation from 22 to 13 has no significant effect on our method. In fact, the optimization here is slightly better but within reasonable variation for stochastic processes such as training a neural network and variance in sampling locations. We do not claim that fewer points work better but include this experiment to show that our method is not dependent on large amounts of data. We forgo MAML comparisons going forward as this approach empirically struggles with parametric PDEs in the context of PINNs and, specifically, L-BFGS optimization.

5.4. 2D nonlinear Allen-Cahn Equation

We consider the following 2D nonlinear Allen-Cahn equation, which is a widely used model for multi-phase flows:

$$\lambda \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + u \left(u^2 - 1 \right) = f, \, (x, y) \in \Omega \tag{13}$$

where $\Omega = [-1, 1]^2$ and where $\lambda \in (0, \pi]$ represents the mobility and u denotes the order parameter, which denotes the different phases. We specify an exact solution as $u(x, y; \lambda) = exp(-\lambda(x + 0.7)) \, sin(\pi x) \, sin(\pi y)$ and derive the corresponding form of the forcing f. We initially randomly sample our parameter space with 16 PINNs approximations containing 8 hidden layers of width 10.

Table 5: We sample the PDE parameter space uniformly with 32 points to compute the values in the table.

Initialization Methods	Error after Adam (10^{-3})	Error after L-BFGS (10^{-3})	L-BFGS Time (s)
Random	1905.5 ± 1591.1	14.5 ± 11.9	469 ± 161
Center	173.3 ± 172.4	12.3 ± 4.6	201 ± 83
Multitask	43.3 ± 45.7	11.5 ± 4.7	120 ± 44
LMC	57.2 ± 44.6	11.4 ± 4.0	120 ± 21
Spline	27.6 ± 35.8	11.4 ± 4.0	63 ± 35
RBF (cubic)	21.9 ± 20.1	11.6 ± 4.3	64 ± 42
RBF (gaussian)	105.0 ± 162.1	12.7 ± 5.2	159 ± 103
RBF (multiquadric)	19.1 ± 13.5	12.0 ± 5.3	68 ± 46
Polynomial	30.6 ± 36.7	11.5 ± 4.2	44 ± 19

In Table 5 we observe similar trends to the previous two equations. However, we can see the problem is substantially more difficult based on the post L-BFGS accuracy and cost. We again note that by simply initializing with the center run, we gain considerable improvement, regardless of the difficulty of the problem.

5.5. 2D nonlinear Diffusion-Reaction Equation

We consider the following 2D nonlinear diffusion-reaction equation:

$$\lambda \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + k \left(u^2 \right) = f, \, (x, y) \in \Omega \tag{14}$$

where $\Omega = [-1, 1]^2$. Here $\lambda \in [1, \pi]$ represents the diffusion coefficient and $k \in [1, \pi]$ represents the reaction rate and f denotes the source term. We specify an exact solution as $u(x, y; \lambda, k) = k \, sin(\pi x) \, sin(\pi y) \, exp(-\lambda \sqrt{(kx^2 + y^2)})$ and derive the corresponding form of the forcing f. We initially randomly sample our parameter space with 22 PINNs approximations containing 8 hidden layers of width 20.

Table 6: We sample the PDE parameter space uniformly with 64 points to compute the values in the table.

Initialization Methods	Error after Adam (10^{-3})	Error after L-BFGS (10^{-3})	L-BFGS Time (s)
Random	581.1 ± 380.2	10.6 ± 6.2	1073 ± 206
Center	92.5 ± 94.9	9.5 ± 5.9	426 ± 159
Multitask	25.4 ± 24.6	9.0 ± 5.4	243 ± 93
LMC	38.0 ± 31.5	9.1 ± 5.8	302 ± 127
Spline	137.1 ± 603.8	8.2 ± 4.9	403 ± 260
RBF (cubic)	39.4 ± 70.7	8.3 ± 4.7	318 ± 217
RBF (gaussian)	246.3 ± 374.7	8.9 ± 5.0	680 ± 406
RBF (multiquadric)	30.3 ± 66.9	8.1 ± 4.6	280 ± 161
Polynomial	13.4 ± 8.7	8.5 ± 5.1	249 ± 129

In Table 6 we start to see the interpolation methods come closer to the improvement provided by the center initialization. They both still significantly improve over a randomized initialization. Another trend to note is that in the "easier" problems, the best methods were the traditional ones; Spline, RBF, and polynomial. Now we observe that Multitask GPs and LMC start to work as well, if not better.

5.6. 10D Equation

We consider the following 10D equation:

$$-\nabla \cdot [\nabla u(1 - \frac{1}{2\pi}\sum_{j=1}^{m}\frac{\boldsymbol{\xi}^{(j)}}{j}cos(j(x+y)))] = f,\ (x,y) \in \Omega, \quad \boldsymbol{\xi} = (\xi^{(1)}, ..., \xi^{(m)}) \tag{15}$$

where $\Omega = [-1,1]^2$, $\boldsymbol{\xi} = [0,2]^{10}$, and m = 10. We specify an exact solution as $u(x,y;\boldsymbol{\xi}) = e^{-\sqrt{\boldsymbol{\xi}^{(1)}x^2+\boldsymbol{\xi}^{(2)}y^2}}sin(\pi x)+e^{-\sqrt{\boldsymbol{\xi}^{(3)}x^2+\boldsymbol{\xi}^{(4)}y^2}}\frac{sin(\pi y)}{4}+e^{-\sqrt{\boldsymbol{\xi}^{(5)}x^2+\boldsymbol{\xi}^{(6)}y^2}}\frac{cos(\pi x)}{8}+ e^{-\sqrt{\boldsymbol{\xi}^{(7)}x^2+\boldsymbol{\xi}^{(8)}y^2}}\frac{cos(\pi y)}{16}+e^{-\sqrt{\boldsymbol{\xi}^{(9)}x^2+\boldsymbol{\xi}^{(10)}y^2}}\frac{sin(\pi(x+y))}{32}$ and derive the corresponding form of the forcing f. The form of the PDE is analogous to a diffusion equation where the coefficient is the first 10 terms of a Fourier expansion.

One caveat is that the coefficient term must be positive for m, in the case of $m = 10$ and $\boldsymbol{\xi} = [0,2]^m$, $|1 - \frac{1}{2\pi}\sum_{j=1}^{10}\frac{\boldsymbol{\xi}^{(j)}}{j}cos(j(x+y))| > 1 - \frac{1}{\pi}\sum_{j=1}^{10}\frac{1}{j} = 0.068$. The form of the PDE and the exact solution is such that the intrinsic dimension is low by reducing the importance of additional PDE parameters.

Table 7: We sample the PDE parameter space with 100 LHS points to compute the values in the table.

Initialization Methods	Error after Adam (10^{-3})	Error after L-BFGS (10^{-3})	L-BFGS Time (s)
Random	73.9 ± 26.3	2.6 ± 2.0	1762 ± 516
Center	16.9 ± 11.0	1.7 ± 0.8	1095 ± 466
Multitask	13.3 ± 8.4	1.8 ± 1.0	921 ± 434
LMC	21.0 ± 17.7	1.6 ± 0.9	962 ± 425
RBF (multiquadric)	23.7 ± 42.1	2.1 ± 2.1	1039 ± 512
Polynomial	12.4 ± 10.4	1.6 ± 0.8	973 ± 468

In Table 7 we sub-select to the best performing RBF method and do not perform spline interpolation as it is non-trivial for a 10D problem. The difference between interpolation and center initialization is minimal, but both vastly outperform randomization.

Further investigation is necessary into quantifying how different parametric PDEs and domains affect the interpolating methods, particularly at high dimensions.

In our observation, the most promising method in terms of cost-benefit is initializing with a run at the center of the parameter domain. However, for the lower-dimensional problems, we do note that interpolating can achieve 10^{-3} error without any training, which is very promising. As the dimension and complexity grow, center initialization seems to be the most enticing method. The cost is negligible to do only one run, and in all cases, it is a vast improvement over randomized weights. Center initialization is easily implemented, and we believe it has use whenever working with parameterized PDEs in which one can exploit the similarity in weights in the parametric domain. Further investigation is warranted in regards to identifying task boundaries in the parametric space, so that this method can be used without prior knowledge of the problem. In this paper, we assume constant task regions as elaborated on in Section 2.3.

6. Summary and Conclusions

In this paper, we have successfully metalearned weight initializations of Physics-informed Neural Networks (PINNs) [1, 2, 3] using a model-aware approach for parametric PDE problems. Having summarized the metalearning approach, along with the collocating version of PINNs, we gather data using initializations from a fully optimized PINN at the center of the parameter domain. By exploiting the smoothness of parametric PDEs in the weight domain when initialized in this way, we can interpolate optimal weight initialization. Using the data collected, we employ a survey of interpolation methods and empirically show they provide weight initializations that greatly improve optimization. We also compare to the standard model-agnostic metalearning method (MAML)[25]. Future investigations will explore the problem of task regions and how to approach metalearning for more complex parametric domains.

Appendix A. Symbols and Notations

Table A.8: Symbols and Notations

u	PDE solution
$\tilde{u}$	PINN approximation of PDE solution
m	PDE parameter dimension
$\mathcal{X}$	Parameter domain, $\mathcal{X} \subset \mathbb{R}^m$
$\boldsymbol{\xi}$	Parameter value, $\boldsymbol{\xi} \in \mathcal{X}$
$\boldsymbol{\xi}^{\mathcal{C}}$	Centroid of $\mathcal{X}$
K	Number of sample task parameters, $\{\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_K\} \subset \mathbb{R}^m$
$\boldsymbol{\xi}_i$	Parameter value in the set, $\boldsymbol{\xi}_i \in \mathcal{X},\ i = 1, ..., K$
$\boldsymbol{\xi}^{(i)}$	Parameter value in the i-dimension, $\boldsymbol{\xi} = \left(\xi^{(i)}, ..., \xi^{(m)}\right)$
$\boldsymbol{\Xi}$	Matrix of parameter values, $\boldsymbol{\Xi} = (\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_K)$
s	Spatial dimension
Ω	Spatial domain of interest, $\Omega \subset \mathbb{R}^p$
$\boldsymbol{x}$	Spatial value, $\boldsymbol{x} \in \Omega$
T	Temporal domain of interest
t	Temporal value, $t \in [0, T]$
ℓ	Number of hidden layers
M_ℓ	Number of trainable weights
$\underline{w}$	Trained PINN weight vector
$\underline{\hat{w}}$	Weight predictor, $\underline{\hat{w}} : \boldsymbol{\xi} \to \underline{w}$
d	Order of hyperbolic cross

Acknowledgements: The authors would like to acknowledge helpful discussions with Professor George Karniadakis and his group (Brown University). This work was funded under AFOSR MURI FA9550-20-1-0358.

Bibliography

References

[1] M. Raissi, P. Perdikaris, G. 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 (2019) 686–707.

[2] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561.

[3] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, arXiv preprint arXiv:1711.10566.

[4] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, The MIT Press, Cambridge, MA, USA, 2016.

[5] M. Alber, A. Buganza Tepole, W. R. Cannon, S. De, S. Dura-Bernal, K. Garikipati, G. Karniadakis, W. W. Lytton, P. Perdikaris, L. Petzold, E. Kuhl, Integrating machine learning and multiscale modeling perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences, Nature: npg digital medicine 2 (115) 115.

[6] J. Schmidhuber, Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook, Ph.D. thesis, Technische Universität München (1987).

[7] S. Thrun, L. Pratt, Learning to learn, Springer Science & Business Media, 2012.

[8] R. Caruana, Multitask learning., Machine Learning.

[9] S. J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on knowledge and data engineering 22 (10) (2009) 1345–1359.

[10] L. Torrey, J. Shavlik, Transfer learning, in: Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, IGI global, 2010, pp. 242–264.

[11] G. Koch, R. Zemel, R. Salakhutdinov, Siamese neural networks for one-shot image recognition, in: ICML deep learning workshop, Vol. 2, Lille, 2015.

[12] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, D. Wierstra, Matching networks for one shot learning, arXiv preprint arXiv:1606.04080.

[13] J. Snell, K. Swersky, R. S. Zemel, Prototypical networks for few-shot learning, arXiv preprint arXiv:1703.05175.

[14] B. N. Oreshkin, P. Rodriguez, A. Lacoste, Tadam: Task dependent adaptive metric for improved few-shot learning, arXiv preprint arXiv:1805.10123.

[15] K. Allen, E. Shelhamer, H. Shin, J. Tenenbaum, Infinite mixture prototypes for few-shot learning, in: International Conference on Machine Learning, PMLR, 2019, pp. 232–241.

[16] S. Hochreiter, A. S. Younger, P. R. Conwell, Learning to learn using gradient descent, in: International Conference on Artificial Neural Networks, Springer, 2001, pp. 87–94.

[17] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. De Freitas, Learning to learn by gradient descent by gradient descent, arXiv preprint arXiv:1606.04474.

[18] K. Li, J. Malik, Learning to optimize, arXiv preprint arXiv:1606.01885.

[19] S. Ravi, H. Larochelle, Optimization as a model for few-shot learning, in: In International Conference on Learning Representations (ICLR), 2017.

[20] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, T. Lillicrap, Meta-learning with memory-augmented neural networks, in: International conference on machine learning, PMLR, 2016, pp. 1842–1850.

[21] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, P. Abbeel, Rl2: Fast reinforcement learning via slow reinforcement learning, arXiv preprint arXiv:1611.02779.

[22] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, M. Botvinick, Learning to reinforcement learn, arXiv preprint arXiv:1611.05763.

[23] T. Munkhdalai, H. Yu, Meta networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 2554–2563.

[24] N. Mishra, M. Rohaninejad, X. Chen, P. Abbeel, A simple neural attentive meta-learner, arXiv preprint arXiv:1707.03141.

[25] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 1126–1135.

[26] C. Finn, Learning to learn with gradients, Ph.D. thesis, UC Berkeley (2018).

[27] L. Bertinetto, J. F. Henriques, P. H. Torr, A. Vedaldi, Meta-learning with differentiable closed-form solvers, arXiv preprint arXiv:1805.08136.

[28] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, S. Whiteson, Fast context adaptation via meta-learning, in: International Conference on Machine Learning, PMLR, 2019, pp. 7693–7702.

[29] Z. Li, F. Zhou, F. Chen, H. Li, Meta-sgd: Learning to learn quickly for few-shot learning, arXiv preprint arXiv:1707.09835.

[30] C. Finn, K. Xu, S. Levine, Probabilistic model-agnostic meta-learning, arXiv preprint arXiv:1806.02817.

[31] F. Zhou, B. Wu, Z. Li, Deep meta-learning: Learning to learn in the concept space, arXiv preprint arXiv:1802.03596.

[32] J. Harrison, A. Sharma, M. Pavone, Meta-learning priors for efficient online bayesian regression, in: International Workshop on the Algorithmic Foundations of Robotics, Springer, 2018, pp. 318–337.

[33] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, R. Hadsell, Meta-learning with latent embedding optimization, arXiv preprint arXiv:1807.05960.

[34] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, et al., Meta-dataset: A dataset of datasets for learning to learn from few examples, arXiv preprint arXiv:1903.03096.

[35] T. Hospedales, A. Antoniou, P. Micaelli, A. Storkey, Meta-learning in neural networks: A survey, arXiv preprint arXiv:2004.05439.

[36] M. Barrault, Y. Maday, N. C. Nguyen, A. T. Patera, An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations, 339 (2004) 667–672.

[37] D. Higdon, J. Gattiker, B. Williams, M. Rightley, Computer model calibration using high-dimensional output, Journal of the American Statistical Association 103 (2008) 570–583.

[38] J. S. Hesthaven, G. Rozza, B. Stamm, Certified Reduced Basis Methods for Parameterized Partial Differential Equations, Springer International Publishing, 2016.

[39] P. Perdikaris, M. Raissi, A. Damianou, N. Lawrence, G. Karniadakis, Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 473 (2017) 20160751.

[40] K. Cutajar, M. Pullin, A. Damianou, N. Lawrence, J. González, Deep Gaussian processes for multi-fidelity modeling, arXiv preprint arXiv:1903.07320.

[41] W. Xing, R. M. Kirby, S. Zhe, Deep Coregionalization for the Emulation of Simulation-Based Spatial-Temporal Fields, Journal of Computational Physics 428 (2021) 109984.

[42] V. Shankar, G. B. Wright, A. L. Fogelson, R. M. Kirby, A radial basis function (rbf)-finite difference method for the simulation of reaction-diffusion equations on stationary platelets within the augmented forcing method, International Journal for Numerical Methods in Fluids 75 (2014) 1–22.

[43] V. Shankar, G. B. Wright, R. M. Kirby, A. L. Fogelson, A radial basis function (rbf)-finite difference (fd) method for diffusion and reaction-diffusion equations on surfaces, Journal of Scientific Computing 63 (2014) 745–768.

[44] Y. Shin, J. Darbon, G. E. Karnaidakis, On the convergence of physics-informed neural networks for linear second order elliptic and parabolic type PDEs, Communications in Computational Physics 28 (2020) 2042.

[45] A. D. Jagtap, E. Kharazmi, G. E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forwar Computer Methods in Applied Mechanics and Engineering 365 (2020) 113028. doi:https://doi.org/10.1016/j.cma.2020.113028.
URL https://www.sciencedirect.com/science/article/pii/S0045782520302127

[46] E. Kharazmi, Z. Zhang, G. E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations (2019). arXiv:1912.00873.

[47] X. Meng, Z. Li, D. Zhang, G. E. Karniadakis, Ppinn: Parareal physics-informed neural network for time-dependent pdes, Computer Methods in Applied Mechanics and Engineering 370 (2020) 113250. doi:https://doi.org/10.1016/j.cma.2020.113250.
URL https://www.sciencedirect.com/science/article/pii/S0045782520304357

[48] D. Zhang, L. Lu, L. Guo, G. E. Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic probl

Journal of Computational Physics 397 (2019) 108850. doi:https://doi.org/10.1016/j.jcp.2019.07.048.
URL https://www.sciencedirect.com/science/article/pii/S0021999119305340

[49] G. Pang, L. Lu, G. E. Karniadakis, fpinns: Fractional physics-informed neural networks (2018). arXiv:1811.08967.

[50] X. I. A. Yang, S. Zafar, J.-X. Wang, H. Xiao, Predictive large-eddy-simulation wall modeling via physics-informed neural networks, Phys. Rev. Fluids 4 (2019) 034602. doi:10.1103/PhysRevFluids.4.034602.
URL https://link.aps.org/doi/10.1103/PhysRevFluids.4.034602

[51] G. Pang, M. D'Elia, M. Parks, G. E. Karniadakis, npinns: nonlocal physics-informed neural networks for a parametrized nonlocal universal laplacian operator. algorithms and applications (2020). arXiv:2004.04276.

[52] A. Jagtap, G. 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 (2020) 2002–2041. doi:10.4208/cicp.OA-2020-0164.

[53] A. Cohen, R. DeVore, Approximation of high-dimensional parametric PDEs, Acta Numerica 24 (2015) 1–159. doi:10.1017/S0962492915000033.

[54] A. Cohen, R. DeVore, C. Schwab, Convergence Rates of Best N-term Galerkin Approximations for a Class of Elliptic sPDEs, Foundations of Computational Mathematics 10 (6) (2010) 615–646. doi:10.1007/s10208-010-9072-2.

[55] C. E. Rasmussen, C. K. I. Williams, Gaussian Processes for Machine Learning, MIT Press, 2006.

[56] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, A. G. Wilson, Gpytorch: Blackbox matrix-matrix gaussian process inference with GPU acceleration, CoRR abs/1809.11165. arXiv:1809.11165.
URL http://arxiv.org/abs/1809.11165

[57] C. Williams, E. V. Bonilla, K. M. Chai, Multi-task gaussian process prediction, Advances in neural information processing systems (2007) 153–160.

[58] A. G. Journel, C. J. Huijbregts, Mining geostatistics, Vol. 600, Academic press London, 1978.

[59] C. d. Boor, A Practical Guide to Splines, Applied Mathematical Sciences, Springer-Verlag, New York, 1978.

[60] C. Lemieux, Monte Carlo and Quasi-Monte Carlo Sampling, Springer Series in Statistics, Springer New York, New York, NY, 2009. doi:10.1007/978-0-387-78165-5.

[61] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272. doi:10.1038/s41592-019-0686-2.

[62] G. F. Fasshauer, Meshfree Approximation Methods With Matlab, World Scientific Publishing Company, Singapore ; Hackensack, N.J, 2007.

[63] D. Dng, V. Temlyakov, T. Ullrich, Hyperbolic Cross Approximation, Springer, 2018, google-Books-ID: xQJ2DwAAQBAJ.

[64] D. Dng, M. Griebel, Hyperbolic cross approximation in infinite dimensions, Journal of Complexity 33 (2016) 55–88. doi:10.1016/j.jco.2015.09.006.
URL https://www.sciencedirect.com/science/article/pii/S0885064X15001016

[65] A. Chkifa, N. Dexter, H. Tran, C. Webster, Polynomial approximation via compressed sensing of high-dimensional functions on lower sets, Mathematics of Computation 87 (311) (2018) 1415–1450. doi:10.1090/mcom/3272.

[66] L. Guo, A. Narayan, L. Yan, T. Zhou, Weighted Approximate Fekete Points: Sampling for Least-Squares Polynomial Approximation, SIAM Journal on Scientific Computing 40 (1) (2018) A366–A387, arXiv:1708.01296 [math.NA]. doi:10.1137/17M1140960.

[67] The SCI Institute, University of Utah, UncertainSCI, https://github.com/SCIInstitute/UncertainSCI (2020).

[68] L. Yang, X. Meng, G. E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse problems with noisy data, Journal of Computational Physics 425 (2021) 109913.

[69] C. Basdevant, M. Deville, P. Haldenwang, J. Lacroix, J. Ouazzani, R. Peyret, P. Orlandi, A. Patera, Spectral and finite difference solutions on the Burgers equation, Computers & Fluids 14 (1986) 23–41.