Estimates on the generalization error of Physics Informed NeuralNetworks (PINNs) for approximating PDEs.

S. Mishra and R. Molinaro

Research Report No. 2020-45
June 2020

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Estimates on the generalization error of Physics Informed Neural Networks (PINNs) for approximating PDEs.

Siddhartha Mishra *and Roberto Molinaro †

June 29, 2020

Abstract

Physics informed neural networks (PINNs) have recently been widely used for robust and accurate approximation of PDEs. We provide rigorous upper bounds on the generalization error of PINNs approximating solutions of the forward problem for PDEs. An abstract formalism is introduced and stability properties of the underlying PDE are leveraged to derive an estimate for the generalization error in terms of the training error and number of training samples. This abstract framework is illustrated with several examples of nonlinear PDEs. Numerical experiments, validating the proposed theory, are also presented.

1 Introduction

Deep learning has emerged as a central tool in science and technology in the last few years. It is based on using deep artificial neural networks (DNNs), which are formed by composing many layers of affine transformations and scalar non-linearities. These deep neural networks have been applied with tremendous success [24] in a variety of tasks such as image and text classification, speech and natural language processing, robotics, game intelligence and protein folding [12], among others.

Partial differential equations (PDEs) model a vast array of natural and manmade phenomena in all areas of science and technology. Explicit solution formulas are only available for very specific types and examples of PDEs. Hence, numerical simulations are necessary for most practical applications featuring PDEs. A diverse set of methods for approximating PDEs numerically are available, such as finite difference, finite element, finite volume and spectral methods. Although very successful in practice, it is still challenging to numerically simulate problems such as Uncertainty quantification (UQ), multi-scale and multi-physics problems, Inverse and constrained optimization problems, PDEs in domains with very complex geometries and PDEs in very high dimensions. Could deep learning assist in the computations of these hitherto difficult to simulate problems involving PDEs?

Deep learning techniques are being increasingly used in the numerical approximations of PDEs. A brief and very incomplete survey of this rapidly emerging literature follows: one approach in using deep neural networks (DNNs) for numerically approximating PDEs is based on explicit (or semi-implicit) representation formulas such as the Feynman-Kac formula for parabolic (and elliptic) PDEs, whose compositional structure is in turn utilized to facilitate approximation by DNNs. This approach is presented and analyzed for a variety of (parametric) elliptic and parabolic PDEs in [3,11,18] and references therein, see a recent paper [21] for a similar approach to approximating linear transport equations with deep neural networks.

Another strategy is to enhance existing numerical methods by adding deep learning inspired modules into them, for instance by learning free parameters of numerical schemes from data [32,41] and references therein.

*Seminar for Applied Mathematics (SAM), D-Math
ETH Zürich, Rämistrasse 101, Zürich-8092, Switzerland
†Seminar for Applied Mathematics (SAM), D-Math
ETH Zürich, Rämistrasse 101, Zürich-8092, Switzerland.

A third approach consists of using deep neural networks to learn *observables* or quantities of interest of the solutions of the underlying PDEs, from data. This approach has been described in [26–28, 34] in the context of uncertainty quantification and PDE constrained optimization and [13] for model order reduction, among others.

Finally, deep neural networks possess the so-called *universal approximation property* [2, 9, 19], namely any continuous, even measurable, function can be approximated by DNNs, see [45] for very precise descriptions of the required neural network architecture for functions with sufficient Sobolev regularity. Hence, it is natural to use deep neural networks as ansatz spaces for the solutions of PDEs, in particular by collocating the PDE residual at training points (see section 2, Algorithm 2.3 for the detailed description of this approach). This approach was first proposed in [22, 23]. However, it has been revived and developed in significantly greater detail in the pioneering contributions of Karniadakis and collaborators, starting with [38, 39]. These authors have termed the underlying neural networks as *Physics Informed Neural Networks* (PINNs) and we will continue to use this, by now, widely accepted terminology in this paper. There has been an explosive growth of papers that present algorithms with PINNs for various applications to both forward and inverse problems for PDEs and a very incomplete list of references include [25, 31, 37, 40] and references therein. Needless to say, PINNs have emerged as a very successful paradigm for approximating different aspects of solutions of PDEs.

Why do PINNs approximate a wide variety of PDEs so well? Although many heuristic reasons have been proposed in some of the afore cited papers, particularly by highlighting the role played by (even small amounts of) data in driving the neural network towards the target and the role of the residual in changing training modes, there is very little rigorous justification of why PINNs work. With the exception of the very recent paper [43], there are few rigorous bounds on the approximation error due to PINNs. In [43], the authors prove consistency of PINNs by showing convergence, under reasonable hypothesis, to solutions of linear elliptic and parabolic PDEs as the number of training samples is increased. A detailed comparison between [43] and the current article is provided in section 6.

The main goal of this paper is to provide a rigorous rationale of why PINNs are so efficient at approximating solutions for the *forward problem* for PDEs, under reasonable and verifiable hypothesis on the underlying PDE. To this end, we will present an abstract framework for PINNs that encompasses a wide variety of potential applications, including to nonlinear PDEs, and prove rigorous estimates on the so-called *generalization error* i.e, the error of the neural network on predicting unseen data. This abstract estimate bounds the generalization error in terms of the underlying training error, number of training (collocation) points and stability bounds for the underlying PDE. Our generalization error estimate will show that the error due to approximating the underlying PDE with a trained PINN will be sufficiently low as long as

- The training error is low i.e, the PINN has been trained well. This error is computed and monitored during the training process. Hence, it is available *a posteriori.*
- The number of training (collocation) points is sufficiently large. This number is determined by the error due to an underlying quadrature rule.
- The solution of the underlying PDE is stable (with respect to perturbations of inputs) in a very precise manner. For nonlinear PDEs, these stability bounds might require that the solutions of the underlying PDEs (and the PINNs) are sufficiently regular.

Thus, with the derived error estimate, we identify possible mechanisms by which PINNs are able to approximate PDEs so well and provide a firm mathematical foundation for approximations by PINNs.

We will also provide three concrete examples to illustrate our abstract framework and error estimate, namely linear and semi-linear parabolic equations, one-dimensional scalar quasilinear parabolic (and hyperbolic) conservation laws and the incompressible Euler equations of fluid dynamics. The abstract error estimate is described in concrete terms for each example and numerical experiments are presented to illustrate and validate the proposed theory. The aim is to convince the reader of why PINNs, when correctly formulated, are successful at approximating the forward problem for PDEs numerically.

The rest of the paper is organized as follows: in section 2, we formulate PINNs for an abstract PDE and prove the estimate on the generalization error. In sections 3, 4 and 5, the abstract framework and error estimate is worked out in the concrete examples of semi-linear Parabolic PDEs, viscous scalar conservation laws and the incompressible Euler equations of fluid dynamics.

2 An abstract framework for Physics informed Neural Networks

2.1 The underlying abstract PDE

Let X, Y be separable Banach spaces with norms $\|\cdot\|_X$ and $\|\cdot\|_Y$, respectively. For definiteness, we set $Y = L^p(\mathbb{D}; \mathbb{R}^m)$ and $X = W^{s,q}(\mathbb{D}; \mathbb{R}^m)$, for $m \geqslant 1$, $1 \leqslant p, q < \infty$ and $s \geqslant 0$, with $\mathbb{D} \subset \mathbb{R}^{\bar{d}}$, for some $\bar{d} \geqslant 1$. In particular, we will also consider space-time domains with $\mathbb{D} = (0,T) \times D \subset \mathbb{R}^d$ with $d \geqslant 1$. In this case $\bar{d} = d + 1$. Let $X^* \subset X$ and $Y^* \subset Y$ be closed subspaces with norms $\|\cdot\|_{X^*}$ and $\|\cdot\|_{Y^*}$, respectively.

We start by considering the following abstract formulation of our underlying PDE:

$$\mathcal{D}(\mathbf{u}) = \mathbf{f}. \tag{2.1}$$

Here, the *differential operator* is a mapping, $\mathcal{D} : X^* \mapsto Y^*$ and the *input* $\mathbf{f} \in Y^*$, such that

$$\begin{aligned} (H1): &\quad \|\mathcal{D}(\mathbf{u})\|_{Y^*} < +\infty, \quad \forall\ \mathbf{u} \in X^*, \text{ with } \|\mathbf{u}\|_{X^*} < +\infty. \\ (H2): &\quad \|\mathbf{f}\|_{Y^*} < +\infty. \end{aligned} \tag{2.2}$$

Moreover, we assume that for all $\mathbf{f} \in Y^*$, there exists a unique $\mathbf{u} \in X^*$ such that (2.1) holds. Furthermore, the solutions of the abstract PDE (2.1) satisfy the following stability bound, let $Z \subset X^* \subset X$ be a closed subspace with norm $\|\cdot\|_Z$.

Assumption 2.1. *For any* $\mathbf{u}, \mathbf{v} \in Z$*, the differential operator* $\mathcal{D}$ *satisfies*

$$(H3): \quad \|\mathbf{u} - \mathbf{v}\|_X \leqslant C_{pde}\left(\|\mathbf{u}\|_Z, \|\mathbf{v}\|_Z\right) \|\mathcal{D}(\mathbf{u}) - \mathcal{D}(\mathbf{v})\|_Y. \tag{2.3}$$

Here, the constant $C_{pde} > 0$ explicitly depends on $|\mathbf{u}\|_Z$ and $\|\mathbf{v}\|_Z$.

As a first example of a PDE with solutions satisfying (H3) (2.3), consider the linear differential operator $\mathcal{D} : X \mapsto Y$ i.e $\mathcal{D}(\alpha\mathbf{u} + \beta\mathbf{v}) = \alpha\mathcal{D}(\mathbf{u}) + \beta\mathcal{D}(\mathbf{v})$, for any $\alpha, \beta \in \mathbb{R}$. For simplicity, let $X^* = X$ and $Y^* = Y$. By the assumptions on the existence and uniqueness of the underlying linear PDE (2.1), there exists an *inverse* operator $\mathcal{D}^{-1} : Y \mapsto X$. Note that the assumption (2.3) is satisfied if the inverse is bounded i.e, $\|\mathcal{D}^{-1}\| \leqslant C < +\infty$, with respect to the natural norm on linear operators from Y to X. Thus, the assumption (2.3) on stability boils down to the boundedness of the inverse operator for linear PDEs. Many well-known linear PDEs possess such bounded inverses [10].

As a second example, we will consider a nonlinear PDE (2.1), but with a well-defined linearization i.e, there exists an operator $\overline{\mathcal{D}} : X^* \mapsto Y^*$, such that

$$\mathcal{D}(\mathbf{u}) - \mathcal{D}(\mathbf{v}) = \overline{\mathcal{D}}_{(\mathbf{u},\mathbf{v})}\left(\mathbf{u} - \mathbf{v}\right), \quad \forall \mathbf{u}, \mathbf{v} \in X^*. \tag{2.4}$$

Again for simplicity, we will assume that $X^* = X$ and $Y^* = Y$. We further assume that the inverse of $\overline{\mathcal{D}}$ exists and is bounded in the following manner,

$$\|\left(\overline{\mathcal{D}}_{(\mathbf{u},\mathbf{v})}\right)^{-1}\| \leqslant C\left(\|\mathbf{u}\|_X, \|\mathbf{v}\|_X\right) < +\infty, \quad \forall \mathbf{u}, \mathbf{v} \in X, \tag{2.5}$$

with the norm of $\overline{\mathcal{D}}^{-1}$ being an operator norm, induced by linear operators from Y to X. Then a straightforward calculation shows that (2.5) suffices to establish the stability bound (2.3).

Further and more concrete examples of PDEs satisfying the above assumptions will be provided in the following sections.

Remark 2.2. We note that initial and boundary conditions are implicitly included in the PDE (2.1). Moreover, by assuming that $Y = L^p(\mathbb{D})$, we are implicitly assuming some regularity for the solutions of the abstract PDE (2.1). In particular, depending on the order of derivatives in the differential operator $\mathcal{D}$, we can expect that the solution $\mathbf{u}$ satisfies the PDE (2.1) in a classical sense i.e, pointwise. ■

2.2 Quadrature rules

In the following section, we need to consider approximating integrals of functions. Hence, we need an abstract formulation for quadrature. To this end, we consider a mapping $g : \mathbb{D} \mapsto \mathbb{R}^m$, such that $g \in Z^* \subset Y^*$. We are interested in approximating the integral,

$$\overline{g} := \int\limits_{\mathbb{D}} g(y) dy,$$

with dy denoting the $\bar{d}$-dimensional Lebesgue measure. In order to approximate the above integral by a quadrature rule, we need the quadrature points $y_i \in \mathbb{D}$ for $1 \leqslant i \leqslant N$, for some $N \in \mathbb{N}$ as well as weights w_i, with $w_i \in \mathbb{R}_+$. Then a quadrature is defined by,

$$\overline{g}_N := \sum_{i=1}^{N} w_i g(y_i), \tag{2.6}$$

for weights w_i and quadrature points y_i. We further assume that the quadrature error is bounded as,

$$\left|\overline{g} - \overline{g}_N\right| \leqslant C_{quad}\left(\|g\|_{Z^*}, \bar{d}\right) N^{-\alpha}, \tag{2.7}$$

for some $\alpha > 0$.

As long as the domain $\mathbb{D}$ is in reasonably low dimension i.e $\bar{d} \leqslant 4$, we can use standard (composite) Gauss quadrature rules on an underlying grid. In this case, the quadrature points and weights depend on the order of the quadrature rule [44] and the rate α depends on the regularity of the underlying integrand i.e, on the space Z^*.

On the other hand, these grid based quadrature rules are not suitable for domains in high dimensions. For moderately high dimensions i.e $4 \leqslant \bar{d} \approx 20$, we can use *low discrepancy sequences*, such as the Sobol and Halton sequences, as quadrature points [6]. As long as the integrand g is of bounded *Hardy-Krause variation* [36], the error in (2.7) converges at a rate $(\log(N))^{\bar{d}} N^{-1}$. One can also employ sparse grids and Smolyak quadrature rules [5] in this regime.

For problems in very high dimensions $\bar{d} \gg 20$, Monte-Carlo quadrature is the numerical integration method of choice [6]. In this case, the quadrature points are randomly chosen, independent and identically distributed (with respect to a scaled Lebesgue measure). The estimate (2.7) holds in the root mean square (RMS) sense and the rate of convergence is $\alpha = \frac{1}{2}$.

2.3 PINNs

In this section, we will describe physics-informed neural networks (PINNs). We start with a description of neural networks which form the basis of PINNs.

2.3.1 Neural Networks.

Given an input $y \in \mathbb{D}$, a feedforward neural network (also termed as a multi-layer perceptron), shown in figure 1, transforms it to an output, through a layer of units (neurons) which compose of either affine-linear maps between units (in successive layers) or scalar non-linear activation functions within units [17], resulting in the representation,

$$\mathbf{u}_\theta(y) = C_K \circ \sigma \circ C_{K-1} \ldots\ldots\ldots \circ \sigma \circ C_2 \circ \sigma \circ C_1(y). \tag{2.8}$$

Here, $\circ$ refers to the composition of functions and σ is a scalar (non-linear) activation function. A large variety of activation functions have been considered in the machine learning literature [17]. Popular choices for the activation function σ in (2.8) include the sigmoid function, the hyperbolic tangent function and the *ReLU* function.

For any $1 \leqslant k \leqslant K$, we define

$$C_k z_k = W_k z_k + b_k, \quad \text{for } W_k \in \mathbb{R}^{d_{k+1} \times d_k}, z_k \in \mathbb{R}^{d_k}, b_k \in \mathbb{R}^{d_{k+1}}. \tag{2.9}$$

For consistency of notation, we set $d_1 = \bar{d}$ and $d_K = m$.

Thus in the terminology of machine learning (see also figure 1), our neural network (2.8) consists of an input layer, an output layer and $(K-1)$ hidden layers for some $1 < K \in \mathbb{N}$. The k-th hidden layer (with d_k neurons) is given an input vector $z_k \in \mathbb{R}^{d_k}$ and transforms it first by an affine linear map C_k (2.9) and then by a nonlinear (component wise) activation σ. A straightforward addition shows that our network contains $\left(\bar{d} + m + \sum\limits_{k=2}^{K-1} d_k\right)$ neurons. We also denote,

$$\theta = \{W_k, b_k\},\ \theta_W = \{W_k\} \quad \forall\ 1 \leqslant k \leqslant K, \tag{2.10}$$

to be the concatenated set of (tunable) weights for our network. It is straightforward to check that $\theta \in \Theta \subset \mathbb{R}^M$ with

$$M = \sum_{k=1}^{K-1} (d_k + 1) d_{k+1}. \tag{2.11}$$

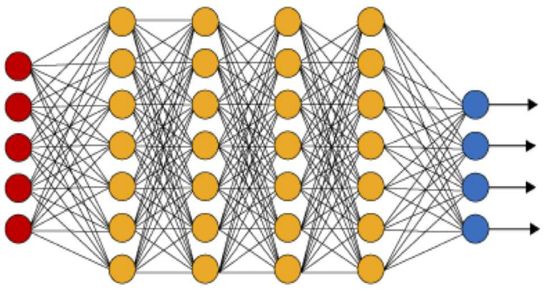


Figure 1: An illustration of a (fully connected) deep neural network. The red neurons represent the inputs to the network and the blue neurons denote the output layer. They are connected by hidden layers with yellow neurons. Each hidden unit (neuron) is connected by affine linear maps between units in different layers and then with nonlinear (scalar) activation functions within units.

2.3.2 Training PINNs: Loss functions and optimization

The neural network $\mathbf{u}_\theta$ (2.8) depends on the tuning parameter $\theta \in \Theta$ of weights and biases. Within the standard paradigm of deep learning [17], one *trains* the network by finding tuning parameters θ such that the loss (error, mismatch, regret) between the neural network and the underlying target is minimized. Here, our target is the solution $\mathbf{u} \in X^*$ of the abstract PDE (2.1) and we wish to find the tuning parameters θ such that the resulting neural network $\mathbf{u}_\theta$ approximates $\mathbf{u}$.

Following standard practice of machine learning, one obtains training data $\mathbf{u}(y)$, for all $y \in \mathbb{S}$, with training set $\mathbb{S} \subset \mathbb{D}$ and then minimizes a loss function of the form $\sum\limits_{\mathbb{S}} \|\mathbf{u}(y) - \mathbf{u}_\theta(y)\|_X$ to find the neural network approximation for $\mathbf{u}$. However, obtaining this training data requires possibly expensive numerical simulations of the underlying PDE (2.1). In order to circumvent this issue, the authors of [23] suggest a different strategy. An abstract paraphrasing of this strategy runs as follows: we assume that for every $\theta \in \Theta$, the neural network $\mathbf{u}_\theta \in X^*$ and $\|\mathbf{u}_\theta\|_{X^*} < +\infty$. We define the following *residual*:

$$\mathcal{R}_\theta = \mathcal{R}(\mathbf{u}_\theta) := \mathcal{D}(\mathbf{u}_\theta) - \mathbf{f}. \tag{2.12}$$

By assumptions (H1),(H2) (2.2), we see that $\mathcal{R} \in Y^*$ and $\|\mathcal{R}\|_{Y^*} < +\infty$ for all $\theta \in \Theta$. Note that $\mathcal{R}(\mathbf{u}) = \mathcal{D}(\mathbf{u}) - \mathbf{f} \equiv 0$, for the solution $\mathbf{u}$ of the PDE (2.1). Hence, the term *residual* is justified for (2.12).

The strategy of PINNs, following [23], is to minimize the *residual* (2.12), over the admissible set of tuning parameters $\theta \in \Theta$ i.e

$$\text{Find } \theta^* \in \Theta : \quad \theta^* = \arg\min_{\theta \in \Theta} \|\mathcal{R}_\theta\|_Y. \tag{2.13}$$

Realizing that $Y = L^p(\mathbb{D})$ for some $1 \leqslant p < \infty$, we can equivalently minimize,

$$\text{Find } \theta^* \in \Theta : \quad \theta^* = \arg\min_{\theta \in \Theta} \|\mathcal{R}_\theta\|^p_{L^p(\mathbb{D})} = \arg\min_{\theta \in \Theta} \int\limits_{\mathbb{D}} |\mathcal{R}_\theta(y)|^p dy. \tag{2.14}$$

As it will not be possible to evaluate the integral in (2.14) exactly, we need to approximate it numerically by a quadrature rule. To this end, we use the quadrature rules (2.6) discussed earlier and select the *training set* $\mathcal{S} = \{y_n\}$ with $y_n \in \mathbb{D}$ for all $1 \leqslant n \leqslant N$ as the quadrature points for the quadrature rule (2.6) and consider the following *loss function*:

$$J(\theta) := \sum_{n=1}^{N} w_n |\mathcal{R}_\theta(y_n)|^p = \sum_{n=1}^{N} w_n \, |\mathcal{D}(\mathbf{u}_\theta(y_n)) - \mathbf{f}(y_n)|^p \,. \tag{2.15}$$

It is common in machine learning [17] to regularize the minimization problem for the loss function i.e we seek to find,

$$\theta^* = \arg\min_{\theta \in \Theta} \left(J(\theta) + \lambda_{reg} J_{reg}(\theta) \right). \tag{2.16}$$

Here, $J_{reg} : \Theta \to \mathbb{R}$ is a *regularization* (penalization) term. A popular choice is to set $J_{reg}(\theta) = \|\theta_W\|_q^q$ for either $q = 1$ (to induce sparsity) or $q = 2$. The parameter $0 \leqslant \lambda_{reg} \ll 1$ balances the regularization term with the actual loss J (2.15).

The above minimization problem amounts to finding a minimum of a possibly non-convex function over a subset of $\mathbb{R}^M$ for possibly very large M. We will follow standard practice in machine learning and solve this minimization problem approximately by either (first-order) stochastic gradient descent methods such as ADAM [20] or even higher-order optimization methods such as LBFGS [14].

For notational simplicity, we denote the (approximate, local) minimum in (2.16) as θ^* and the underlying deep neural network $\mathbf{u}^* = \mathbf{u}_{\theta^*}$ will be our physics-informed neural network (PINN) approximation for the solution $\mathbf{u}$ of the PDE (2.1).

The proposed algorithm for computing this PINN is given below,

Algorithm 2.3. Finding a physics informed neural network to approximate the solution of the PDE (2.1).

Inputs: *Underlying domain $\mathbb{D}$, differential operator $\mathcal{D}$ and input source term $\mathbf{f}$ for the PDE* (2.1)*, quadrature points and weights for the quadrature rule* (2.6)*, non-convex gradient based optimization algorithms.*

Goal: *Find PINN $\mathbf{u}^* = \mathbf{u}_{\theta^*}$ for approximating the PDE* (2.1).

Step 1: *Choose the training set $\mathcal{S} = \{y_n\}$ for $y_n \in \mathbb{D}$, for all $1 \leqslant n \leqslant N$ such that $\{y_n\}$ are quadrature points for the underlying quadrature rule* (2.6).

Step 2: *For an initial value of the weight vector $\overline{\theta} \in \Theta$, evaluate the neural network $\mathbf{u}_{\overline{\theta}}$* (2.8)*, the PDE residual* (2.12)*, the loss function* (2.16) *and its gradients to initialize the underlying optimization algorithm.*

Step 3: *Run the optimization algorithm till an approximate local minimum θ^* of* (2.16) *is reached. The map $\mathbf{u}^* = \mathbf{u}_{\theta^*}$ is the desired PINN for approximating the solution $\mathbf{u}$ of the PDE* (2.1).

Remark 2.4. The standard practice in machine learning is to approximate the solution $\mathbf{u}$ of (2.1) from training data $(z_j, \mathbf{u}(z_j))$, for $z_j \in \mathbb{D}$ and $1 \leqslant j \leqslant N_d$, then one would like to minimize the so-called *data loss*:

$$J_d(\theta) := \frac{1}{N_d} \sum_{j=1}^{N_d} |\mathbf{u}(z_j) - \mathbf{u}_\theta(z_j)|^p. \tag{2.17}$$

Comparing the loss functions (2.15) and (2.17) reveals the essence of PINNs, i.e, for PINNs, one does not necessarily need any training data for the solution $\mathbf{u}$ of (2.1), only the residual (2.12) needs to be evaluated, for which knowledge of the differential operator and inputs for (2.1) suffice. Here, we distinguish between initial and boundary data, which is implicitly included into the formulation of the PDE (2.1) and which are necessary for any numerical solution of (2.1), from other types of data, for instance values of the solution $\mathbf{u}$, from the interior of the domain $\mathbb{D}$. Thus, the PINN in this formulation, which is closer in spirit to the original proposal of [23], can be purely thought of as a numerical method for the PDE (2.1). In this form, one can consider PINNs as an example of *unsupervised learning* [29], as no explicit data is necessary.

On the other hand, the authors of [39] and subsequent papers, have added further flexibility to PINNs by also including training data by augmenting the loss function (2.16) with the data loss (2.17) and seeking neural networks with tuning parameters defined by,

$$\theta^* = \arg\min_{\theta\in\Theta} \left(J_d(\theta) + \lambda J(\theta) + \lambda_{reg} J_{reg}(\theta) \right), \tag{2.18}$$

with an additional hyperparameter λ that balances the data loss with the residual. This paradigm has been very successful for inverse problems, where boundary and initial data may not be known [39, 40]. However, for the rest of the paper, we focus on the forward problem and only consider PINNs, trained with algorithm 2.3, that minimize the residual based loss function 2.16, without needing additional data. ■

Remark 2.5. Algorithm 2.3 requires the residual (2.12), for the neural network $\mathbf{u}_\theta$, to be evaluated pointwise for every training step. Hence, one needs the neural network to be sufficiently regular. Depending on the order of derivatives in $\mathcal{D}$ of (2.1), this can be ensured by requiring sufficient smoothness for the activation function. Hence, the ReLU activation function, which is only Lipschitz continuous, might not be admissible in this framework. On the other hand, smooth activation functions such as logistic and tanh are always admissible. ■

2.4 An abstract estimate on the generalization error

In this section, we will estimate the error due to the PINN (generated by algorithm 2.3) in approximating the solution $\mathbf{u}$ of the PDE (2.1). The relevant concept of error is the so-called *generalization error* (see [42]):

$$\mathcal{E}_G = \mathcal{E}_G(\theta^*; \mathcal{S}) := \|\mathbf{u} - \mathbf{u}^*\|_X \tag{2.19}$$

Clearly, the generalization error depends on the chosen training set $\mathcal{S}$ and the trained neural network with tuning parameters θ^*, found by algorithm 2.3. However, we will suppress this dependence due to notational convenience. We remark that the generalization error (2.19) is the error emanating from approximating the solution $\mathbf{u}$ of (2.1) by the PINN $\mathbf{u}^*$, generated by the algorithm 2.3.

Note that there is no computation of the generalization error during the training process. On the other hand, we monitor the so-called *training error* given by,

$$\mathcal{E}_T := \left(\sum_{n=1}^{N} w_n |\mathcal{R}_{\theta^*}(y_n)|^p \right)^{\frac{1}{p}} = J(\theta^*)^{\frac{1}{p}}. \tag{2.20}$$

Hence, the training error $\mathcal{E}_T$ can be readily computed, after training has been completed, from the loss function (2.16). The generalization error (2.19) can be estimated in terms of the training error in the following theorem.

Theorem 2.6. *Let $\mathbf{u} \in Z \subset X^*$ be the unique solution of the PDE* (2.1) *and assume that the stability hypothesis* (2.3) *holds. Let $\mathbf{u}^* \in Z \subset X^*$ be the PINN generated by algorithm 2.3, based on the training set $\mathcal{S}$ of quadrature points corresponding to the quadrature rule* (2.6)*. Further assume that the residual $\mathcal{R}_{\theta^*}$, defined in* (2.12)*, be such that $\mathcal{R}_{\theta^*} \in Z^*$ and the quadrature error satisfies* (2.7)*. Then the following estimate on the generalization error holds,*

$$\mathcal{E}_G \leqslant C_{pde}\mathcal{E}_T + C_{pde} C_{quad}^{\frac{1}{p}} N^{-\frac{\alpha}{p}}, \tag{2.21}$$

with constants $C_{pde} = C_{pde}(\|\mathbf{u}\|_Z, \|\mathbf{u}^*\|_Z)$ and $C_{quad} = C_{quad}(\||\mathcal{R}_{\theta^*}|^p\|)$ stemming from (2.3) *and* (2.7)*, respectively.*

Proof. In the following, we denote $\mathcal{R} = \mathcal{R}_{\theta^*}$, the residual (2.12), corresponding to the trained neural network $\mathbf{u}^*$. As $\mathbf{u}$ solves the PDE (2.1) and $\mathcal{R}$ is defined by (2.12), we easily see that,

$$\mathcal{R} = \mathcal{D}(\mathbf{u}^*) - \mathcal{D}(\mathbf{u}). \tag{2.22}$$

Hence, we can directly apply the stability bound (2.3) to yield,

$$\begin{aligned}\mathcal{E}_G &= \|\mathbf{u} - \mathbf{u}^*\|_X \quad \text{(by (2.19))}, \\ &\leqslant C_{pde}\|\mathcal{D}(\mathbf{u}^*) - \mathcal{D}(\mathbf{u})\|_Y \quad \text{(by (2.3))}, \\ &\leqslant C_{pde}\|\mathcal{R}\|_Y \quad \text{(by (2.22))}.\end{aligned}$$

By the fact that $Y = L^p(\mathbb{D})$, the definition of the training error (2.20) and the quadrature rule (2.6), we see that,

$$\|\mathcal{R}\|_Y^p \approx \left(\sum_{n=1}^{N} w_n |\mathcal{R}_{\theta^*}|^p\right) = \mathcal{E}_T^p.$$

Hence, the training error is a quadrature for the residual (2.12) and the resulting quadrature error, given by (2.7) translates to,

$$\|\mathcal{R}\|_Y^p \leqslant \mathcal{E}_T^p + C_{quad}N^{-\alpha}.$$

Therefore,

$$\begin{aligned}\mathcal{E}_G^p &\leqslant C_{pde}^p \|\mathcal{R}\|_Y^p \\ &\leqslant C_{pde}^p \left(\mathcal{E}_T^p + C_{quad}N^{-\alpha}\right) \\ \Rightarrow \quad \mathcal{E}_G &\leqslant C_{pde}\mathcal{E}_T + C_{pde}C_{quad}^{\frac{1}{p}}N^{-\frac{\alpha}{p}},\end{aligned}$$

which is the desired estimate (2.21). □

We term a PINN, generated by the algorithm 2.3 to be *well-trained* if the following condition hold,

$$\mathcal{E}_T \leqslant C_{qaud}^{\frac{1}{p}} N^{-\frac{\alpha}{p}}. \tag{2.23}$$

Thus, a well-trained PINN is one for which the training errors are smaller than the so-called *generalization gap* (given by the rhs of (2.23)) which results from quadrature.

The bound (2.21) leads directly to the following *consistency or convergence* lemma for a well-trained network,

Lemma 2.7. *Assume that for any N (size of the training set $\mathcal{S}$), the PINNs generated by the algorithm 2.3, denoted by $\mathbf{u}_N^*$ are well-trained i.e, they satisfy* (2.23)*. Furthermore, assume that there exists constant $\mathcal{C}$, independent of N such that the following holds,*

$$\max\left\{\|\mathbf{u}_N^*\|_Z, \left\||\mathcal{R}\left(\mathbf{u}_N^*\right)|^p\right\|_Y\right\} \leqslant \mathcal{C} < +\infty, \quad \forall N, \tag{2.24}$$

with $\mathcal{R}$ being defined with respect to the trained PINN $\mathbf{u}_N^*$ by (2.12)*. Then we have,*

$$\lim_{N\to\infty} \mathbf{u}_N^* = \mathbf{u}, \quad \text{in } X \tag{2.25}$$

with $\mathbf{u}$ being the unique solution of the PDE (2.1)*.*

Thus, Lemma 2.7 provides consistency for the approximation of the solution of the PDE (2.1) by PINNs. It can be thought of as a variant of the celebrated *Lax equivalence theorem* of classical numerical analysis as stability (in the form of the conditional stability estimate (2.3)) and accuracy (in the form of small training errors and decay of quadrature errors through (2.7) imply convergence of the PINN to the solution of the PDE (2.1).

Remark 2.8. The estimate (2.21) clearly indicates mechanisms that underpin possible efficient approximation of solutions of PDEs by PINNs as it breaks down the sources of error into the following three parts,

- The PINN has to be well-trained i.e, the training error $\mathcal{E}_T$ has to be sufficiently small. Note that we have no a priori control on the training error but can compute it *a posteriori.*
- The underlying solution and PINNs have to be sufficiently regular such that the residual (2.12) can be approximated to high accuracy by the quadrature rule (2.6). The quadrature error has a constant C_{quad} and a rate α, both dependent on the underlying solution and quadrature rule used. The constant can be computed *a posteriori* and the (worst case) rate is known in advance.
- Finally, the whole estimate (2.21) relies on stability of solutions of the underlying PDE, measured by the stability estimate (2.3) on the differential operator $\mathcal{D}$ and appearing as the constant C_{pde} in (2.21). Thus, the estimate leverages stability of PDEs into efficient approximation by PINNs.

■

Remark 2.9. In addition to the requirement that the approximating PINNs are well-trained (2.23), Lemma 2.7 requires the uniform bound (2.24) on the PINNs. Given the structure of the PINN (2.8), in principle this bound can be explicitly estimated in terms of the weights in (2.8) by successive differentiation. Hence, one can penalize the resulting bound on weights, in a manner analogous to the regularization term in (2.16) and ensure (2.24). However, given the complicated algebraic expressions that result in this process, even for a shallow (one hidden layer) neural network, we will not explore this further but rather verify these assumptions a posteriori. ■

2.4.1 Case of random training points.

The quadrature rule (2.6) (and the quadrature error (2.7)) play an important role in the error estimate (2.21). In principle, as long as the underlying solution (and PINN) are sufficiently regular, one can use standard grid based (composite) Gauss quadrature rules and obtain low quadrature errors, resulting in a large α in the estimate (2.21). However, the constant C_{quad}, depends explicitly on the dimension $\bar{d}$ of $\mathbb{D}$ and suffers from the so-called *curse of dimensionality.* This can be alleviated for moderately high dimensions by using low-discrepancy sequences as the quadrature (and training) points. As long as the solution $\mathbf{u}$ (and PINN $\mathbf{u}^*$) are of bounded Hardy-Krause variation [36], we know that $\alpha = 1$ and the constant $C_{quad} = (\log(N))^{\bar{d}}$ for the resulting Quasi-Monte Carlo quadrature error [6]. Note that the logarithmic correction can be dominated as long $N \approx e^{\bar{d}}$. However, if the underlying dimension is very high, using low-discrepancy sequences might not be efficient, see [34] for a similar observation in the context of standard neural networks.

Hence, for problems in very high dimensions, one has to use random quadrature points as the resulting Monte Carlo quadrature does not suffer from the curse of dimensionality. However, as we use the quadrature points as training points for the PINN in algorithm 2.3, we cannot directly use error estimates for Monte Carlo quadrature (central limit theorem) for approximating the integral in $\|\mathcal{R}_{\theta^*}\|_Y^p$ as $\mathcal{R}_{\theta^*}(y_n)$ in the training error (2.20) could be *correlated* (and are not independent) for different training points y_n. In general, advanced tools from statistical learning theory [8, 42] such as Rademacher complexity, VC dimension etc are used to circumvent these correlations and in practice, lead to large overestimates on the generalization error [1]. Here, we follow the approach of [27] and references therein and adopt a pragmatic framework in estimating the generalization error in terms of the training error.

For simplicity of notation, we set the domain $\mathbb{D} = [0,1]^{\bar{d}}$ and $Y = L^1(\mathbb{D})$ and start by recalling that the points in the training set $\mathcal{S}$ are chosen randomly from domain $\mathbb{D}$, independently and identically distributed with the underlying Lebesgue measure dy. We will identify the training set $\mathcal{S}$ with the vector $\mathbf{S} \in \mathbb{D}^N$, defined by

$$\mathbf{S} = [y_1, y_2, \dots y_N],$$

which is distributed according to the measure $d\mathbf{S} := dy \otimes \cdot \cdot dy$ (a N-fold product Lebesgue measure). We also denote $(\Omega, \Sigma, \mathbb{P})$ as the underlying complete probability space from which random draws are made. Expectation with respect to this underlying probability measure $\mathbb{P}$ is denoted as $\mathbb{E}$

Note that as the training set is randomly chosen, the trained PINN $\mathbf{u}^*$, explicitly depends on the training set $\mathbf{S}$ i.e, $\mathbf{u}^* = \mathbf{u}^*(\mathbf{S})$. Consequently, the generalization and training errors now explicitly depend on the training set i.e,

$$\mathcal{E}_G(\mathbf{S}) = \mathcal{E}_G(\theta^*;\mathbf{S}) := \|\mathbf{u}-\mathbf{u}^*(\mathbf{S})\|_X, \quad \mathcal{E}_T(\mathbf{S}) = \mathcal{E}_T(\theta^*;\mathbf{S}) := \frac{1}{N}\sum_{n=1}^{N} |\mathcal{R}_{\theta^*}(y_n;\mathbf{S})|$$

Next, we follow [27] and define the so-called *cumulative* (average over training sets) generalization and training errors as,

$$\bar{\mathcal{E}}_G = \int\limits_{\mathbb{D}^N} \mathcal{E}_G(\mathbf{S})d\mathbf{S} = \int\limits_{\mathbb{D}^N} \|\mathbf{u}-\mathbf{u}^*(\mathbf{S})\|_X d\mathbf{S}, \tag{2.26}$$

and

$$\bar{\mathcal{E}}_T = \int\limits_{\mathbb{D}^N} \mathcal{E}_T(\mathbf{S})d\mathbf{S}. \tag{2.27}$$

Note that the cumulative errors $\bar{\mathcal{E}}_{G,T}$ are deterministic quantities. In [27] and references therein, one followed standard practice in machine learning and also computed the so-called *validation set*,

$$\mathcal{V} = \{z_j \in \mathbb{D},\ 1 \leqslant j \leqslant N, \quad z_j\ i.i.d\ wrt\ dy\}. \tag{2.28}$$

The validation set is chosen before the start of the training process and is independent of the training sets. We can define the *cumulative validation error* as,

$$\bar{\mathcal{E}}_V = \frac{1}{N}\int_{\mathbb{D}^N} \sum_{j=1}^{N} |\mathcal{R}_{\theta^*}(z_n;\mathbf{S})| d\mathbf{S}. \tag{2.29}$$

We observe that as the set $\mathcal{V}$ is drawn randomly from $\mathbb{D}$ with underlying distribution dy, the cumulative validation error is a random quantity, $\bar{\mathcal{E}}_V = \bar{\mathcal{E}}_V(\omega)$ with $\omega \in \mathbf{\Omega}$. We suppress this ω-dependence for notational convenience. Finally, we introduce the *validation gap*:

$$\mathcal{E}_{TV} := \mathbb{E}\left(|\bar{\mathcal{E}}_T - \bar{\mathcal{E}}_V|\right) := \int\limits_{\Omega} |\bar{\mathcal{E}}_T - \bar{\mathcal{E}}_V(\omega)| d\mathbb{P}(\omega) \tag{2.30}$$

Then, we have the following lemma on estimating the cumulative generalization error,

Lemma 2.10. *Let $\mathbf{u} \in X^* \subset X$ be the unique solution of the PDE* (2.1)*. Let $\mathbf{u}^* = \mathbf{u}_{\theta^*}(\mathbf{S})$ be a PINN generated by algorithm 2.3, with N randomly chosen training points, identified by $\mathbf{S} \in \mathbb{D}^N$. Assume that there exists a constant C_{pde} such that the stability bound* (2.3) *is uniformly satisfied for PINNs, generated by the algorithm 2.3 for every training set $\mathbb{S} \subset \mathbb{D}$, then the cumulative generalization error* (2.26) *is bounded by,*

$$\bar{\mathcal{E}}_G \leqslant C_{pde}\left(\bar{\mathcal{E}}_T + \mathcal{E}_{TV} + \frac{std(|\mathcal{R}_{\theta^*}|)}{\sqrt{N}}\right), \tag{2.31}$$

with cumulative training error $\bar{\mathcal{E}}_T$ (2.27)*, validation gap $\mathcal{E}_{TV}$* (2.30) *and*

$$std(|\mathcal{R}_{\theta^*}|) := \sqrt{\mathbb{E}\left(\int_{\mathbb{D}^N} |\mathcal{R}(z(\omega),\mathbf{S})|d\mathbf{S} - \int_{\mathbb{D}}\int_{\mathbb{D}^N} |\mathcal{R}(z,\mathbf{S})|d\mathbf{S}dz\right)^2} \tag{2.32}$$

Proof. We use the shorthand notation $\mathcal{R}(\mathbf{S}) = \mathcal{R}_{\theta^*}(\mathbf{S})$ for the residual (2.12). By definition of the cumulative generalization error (2.26), we have

$$\begin{aligned}
\bar{\mathcal{E}}_G &= \mathbb{E}\left(\|\mathbf{u}-\mathbf{u}^*\|_X\right) \\
&\leqslant C_{pde}\mathbb{E}\left(\|\mathcal{R}(\mathbf{S})\|_1\right), \quad \text{by (2.3)}, \\
&\leqslant C_{pde}\mathbb{E}\left(|\mathbb{E}\left(\|\mathcal{R}(\mathbf{S})\|_1\right) - \bar{\mathcal{E}}_V + \bar{\mathcal{E}}_V - \bar{\mathcal{E}}_T + \bar{\mathcal{E}}_T|\right) \\
&\leqslant C_{pde}\left(\bar{\mathcal{E}}_T + \mathcal{E}_{TV} + \sqrt{\mathbb{E}\left(|\mathbb{E}(\|\mathcal{R}(\mathbf{S})\|_1) - \bar{\mathcal{E}}_V|^2\right)}\right)
\end{aligned}$$

As the residual evaluated on the validation points is independent, the term inside the square root can be easily estimated in terms of the Monte Carlo quadrature to obtain the desired estimate (2.31). □

We remark that the estimate on the cumulative generalization error (2.31) requires the computation of the so-called validation set. This is standard practice in machine learning, albeit with a smaller validation set than the training set. An alternative approach to obtain bounds on the generalization error for random training points would be to use space-filling arguments for random points [7] and some Hölder regularity of the underlying maps. The resulting bound is directly in the form (2.21) and does not require the computation of a validation gap. However, this bound suffers from the curse of dimensionality and we do not investigate it in detail here.

The error estimates (2.21) (and (2.31)) are for the abstract PDE formulation (2.1) and are meant to illustrate possible mechanisms for low approximation errors with PINNs. A lot of information is implicit in them, for instance the exact form of the function spaces X, X^*, Y, Y^* and (initial) boundary conditions. These will be made explicit in each of the subsequent concrete examples.

3 Semi-linear Parabolic equations

3.1 The underlying PDEs

Let $D \subset \mathbb{R}^d$ be a domain i.e, an open connected bounded set with a C^1 boundary ∂D. We consider the following model semi-linear parabolic equation,

$$\begin{aligned} u_t &= \Delta u + f(u), \quad \forall x \in D \ t \in (0,T), \\ u(x,0) &= \bar{u}(x), \quad \forall x \in D, \\ u(x,t) &= 0, \quad \forall x \in \partial D, \ t \in (0,T). \end{aligned} \tag{3.1}$$

Here, $u_0 \in H^{\bar{s}}(D;\mathbb{R})$ is the initial data, $u \in H^s(((0,T)\times D);\mathbb{R})$ is the solution and $f : \mathbb{R}\times\mathbb{R}$ is the non-linear source (reaction) term. We assume that the non-linearity is globally Lipschitz i.e, there exists a constant C_f (independent of v, w) such that

$$|f(v) - f(w)| \leqslant C_f |v - w|, \quad v, w \in \mathbb{R}. \tag{3.2}$$

In particular, the homogeneous linear heat equation with $f(u) \equiv 0$ and the linear source term $f(u) = c_f u$ are examples of (3.1). Semilinear heat equations with globally Lipschitz nonlinearities arise in several models in biology and finance [3].

The existence, uniqueness and regularity of the semi-linear parabolic equations with Lipschitz non-linearities such as (3.1) can be found in classical textbooks such as [15]. For our purposes here, we will choose $\bar{s}$ sufficiently large such that the initial data $\bar{u} \in C^k(D)$ and we obtain $u \in C^k([0,T]\times D)$, with $k \geqslant 2$ as the classical solution of the semi-linear parabolic equation (3.1).

3.2 PINNs

In order to complete the PINNs algorithm 2.3, we need to specify the training set $\mathbb{S}$ and define the appropriate residual, which we do below.

3.2.1 Training set

Let $D_T = D \times (0,T)$ be the space-time domain. As in section 2, we will choose the training set $\mathbb{S} \subset [0,T]\times\bar{D}$, based on suitable quadrature points. We have to divide the training set into the following three parts,

- Interior training points $\mathbb{S}_{int} = \{y_n\}$ for $1 \leqslant n \leqslant N_{int}$, with each $y_n = (x,t)_n \in D_T$. These points can be the quadrature points, corresponding to a suitable space-time grid-based composite Gauss quadrature rule as long $d \leqslant 3$ or correspond to low-discrepancy sequences for moderately high dimensions or randomly chosen points in very high dimensions. See figure 2 for an example of randomly chosen training points for $d = 1$.

- Spatial boundary training points $\mathbb{S}_{sb} = \{z_n\}$ for $1 \leqslant n \leqslant N_{sb}$ with each $z_n = (x,t)_n$ and each $x_n \in \partial D$. Again the points can be chosen from a grid-based quadrature rule on the boundary, as low-discrepancy sequences or randomly.

- Temporal boundary training points $\mathcal{S}_{tb} = \{x_n\}$, with $1 \leqslant n \leqslant N_{tb}$ and each $x_n \in D$, chosen either as grid points, low-discrepancy sequences or randomly chosen in D.

The full training set is $\mathcal{S} = \mathcal{S}_{int} \cup \mathcal{S}_{sb} \cup \mathcal{S}_{tb}$. An example for the full training set is shown in figure 2.

3.2.2 Residuals

In the algorithm 2.3 for generating PINNs, we need to define appropriate residuals. For the neural network $u_\theta \in C^k([0,T] \times \bar{D})$, with continuous extensions of the derivatives to the boundaries, defined by (2.8), with a smooth activation function such as $\sigma = \tanh$ and $\theta \in \Theta$ as the set of tuning parameters, we define the following residual,

- Interior Residual given by,

$$\mathcal{R}_{int,\theta}(x,t) := \partial_t u_\theta(x,t) - \Delta u_\theta(x,t) - f(u_\theta(x,t)). \tag{3.3}$$

  Here $\Delta = \Delta_x$ is the spatial Laplacian. Note that the residual is well defined and $\mathcal{R}_{int,\theta} \in C^{k-2}([0,T]\times\bar{D})$ for every $\theta \in \Theta$.

- Spatial boundary Residual given by,

$$\mathcal{R}_{sb,\theta}(x,t) := u_\theta(x,t), \quad \forall x \in \partial D,\ t \in (0,T]. \tag{3.4}$$

  Given the fact that the neural network is smooth, this residual is well defined.

- Temporal boundary Residual given by,

$$\mathcal{R}_{tb,\theta}(x) := u_\theta(x,0) - \bar{u}(x), \quad \forall x \in D. \tag{3.5}$$

  Again this quantity is well-defined and $\mathcal{R}_{tb,\theta} \in C^k(D)$ as both the initial data and the neural network are smooth.

3.2.3 Loss function

As in section 2, we need a loss function to train the PINN. To this end, we set the following loss function,

$$J(\theta) := \sum_{n=1}^{N_{tb}} w_n^{tb} |\mathcal{R}_{tb,\theta}(x_n)|^2 + \sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,\theta}(x_n,t_n)|^2 + \lambda \sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{int,\theta}(x_n,t_n)|^2. \tag{3.6}$$

Here the residuals are defined by (3.5), (3.4), (3.3), w_n^{tb} are the N_{tb} quadrature weights corresponding to the temporal boundary training points $\mathcal{S}_{tb}$, w_n^{sb} are the N_{sb} quadrature weights corresponding to the spatial boundary training points $\mathcal{S}_{sb}$ and w_n^{int} are the N_{int} quadrature weights corresponding to the interior training points $\mathcal{S}_{int}$. Furthermore, λ is a hyperparameter for balancing the residuals, on account of the PDE and the initial and boundary data, respectively.

3.3 Estimate on the generalization error.

The algorithm 2.3 for training a PINN to approximate the semilinear parabolic equation (3.1) is now completely specified and can be run to generate the required PINN, which we denote as $u^* = u_{\theta^*}$, where $\theta^* \in \Theta$ is the (approximate) minimizer of the optimization problem, corresponding to the loss function (2.16), (3.6).

We are interested in estimating the generalization error (2.19) for this PINN. In this case, $X = L^2(D \times (0,T))$ and the generalization error is concretely defined as,

$$\mathcal{E}_G := \left(\int_0^T \int_D |u(x,t) - u^*(x,t)|^2 dx dt \right)^{\frac{1}{2}}. \tag{3.7}$$

As for the abstract PDE (2.1), we are going to estimate the generalization error in terms of the *training error* that we define as,

$$\mathcal{E}_T^2 := \underbrace{\sum_{n=1}^{N_{tb}} w_n^{tb} |\mathcal{R}_{tb,\theta^*}(x_n)|^2}_{(\mathcal{E}_T^{tb})^2} + \underbrace{\sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,\theta^*}(x_n,t_n)|^2}_{(\mathcal{E}_T^{sb})^2} + \lambda \underbrace{\sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{int,\theta^*}(x_n,t_n)|^2}_{(\mathcal{E}_T^{int})^2}. \tag{3.8}$$

Note that the training error can be readily computed *a posteriori* from the loss function (2.16),(3.6).

We also need the following assumptions on the quadrature error, analogous to (2.3). For any function $g \in C^k(D)$, the quadrature rule corresponding to quadrature weights w_n^{tb} at points $x_n \in \mathcal{S}_{tb}$, with $1 \leqslant n \leqslant N_{tb}$, satisfies

$$\left| \int_D g(x)dx - \sum_{n=1}^{N_{tb}} w_n^{tb} g(x_n) \right| \leqslant C_{quad}^{tb}(\|g\|_{C^k}) N_{tb}^{-\alpha_{tb}}. \tag{3.9}$$

For any function $g \in C^k(\partial D \times [0,T])$, the quadrature rule corresponding to quadrature weights w_n^{sb} at points $(x_n,t_n) \in \mathcal{S}_{sb}$, with $1 \leqslant n \leqslant N_{sb}$, satisfies

$$\left| \int_0^T \int_{\partial D} g(x,t) ds(x) dt - \sum_{n=1}^{N_{sb}} w_n^{sb} g(x_n,t_n) \right| \leqslant C_{quad}^{sb}(\|g\|_{C^k}) N_{sb}^{-\alpha_{sb}}. \tag{3.10}$$

Finally, for any function $g \in C^\ell(D \times [0,T])$, the quadrature rule corresponding to quadrature weights w_n^{int} at points $(x_n,t_n) \in \mathcal{S}_{int}$, with $1 \leqslant n \leqslant N_{int}$, satisfies

$$\left| \int_0^T \int_D g(x,t) dx dt - \sum_{n=1}^{N_{int}} w_n^{int} g(x_n,t_n) \right| \leqslant C_{quad}^{int}(\|g\|_{C^\ell}) N_{int}^{-\alpha_{int}}. \tag{3.11}$$

In the above, $\alpha_{int}, \alpha_{sb}, \alpha_{tb} > 0$ and in principle, different order quadrature rules can be used. We estimate the generalization error for the PINN in the following,

Theorem 3.1. *Let $u \in C^k(\bar{D} \times [0,T])$ be the unique classical solution of the semilinear parabolic euqation* (3.1) *with the source f satisfying* (3.2). *Let $u^* = u_{\theta^*}$ be a PINN generated by algorithm 2.3, corresponding to loss function* (2.16), (3.6). *Then the generalization error* (3.7) *can be estimated as,*

$$\mathcal{E}_G \leqslant C_1 \left(\mathcal{E}_T^{tb} + \mathcal{E}_T^{int} + C_2 (\mathcal{E}_T^{sb})^{\frac{1}{2}} + (C_{quad}^{tb})^{\frac{1}{2}} N_{tb}^{-\frac{\alpha_{tb}}{2}} + (C_{quad}^{int})^{\frac{1}{2}} N_{int}^{-\frac{\alpha_{int}}{2}} + C_2 (C_{quad}^{sb})^{\frac{1}{4}} N_{sb}^{-\frac{\alpha_{sb}}{4}} \right), \tag{3.12}$$

with constants given by,

$$\begin{aligned} C_1 &= \sqrt{T + (1+2C_f)T^2 e^{(1+2C_f)T}}, \quad C_2 = \sqrt{C_{\partial D}(u,u^*) T^{\frac{1}{2}}}, \\ C_{\partial D} &= |\partial D|^{\frac{1}{2}} \left(\|u\|_{C^1([0,T]\times\partial D)} + \|u^*\|_{C^1([0,T]\times\partial D)} \right), \end{aligned} \tag{3.13}$$

and $C_{quad}^{tb} = C_{quad}^{tb}(\|\mathcal{R}_{tb,\theta^*}\|_{C^k})$, $C_{quad}^{sb} = C_{quad}^{tb}(\|\mathcal{R}_{sb,\theta^*}\|_{C^k})$ and $C_{quad}^{int} = C_{quad}^{int}(\|\mathcal{R}_{int,\theta^*}\|_{C^{k-2}})$ are the constants defined by the quadrature error (3.9), (3.10), (3.11), *respectively.*

Proof. By the definitions of the residuals (3.3), (3.4), (3.5) and the underlying PDE (3.1), we can readily verify that the error $\hat{u} : u^* - u$ satisfies the following (forced) parabolic equation,

$$\begin{aligned} \hat{u}_t &= \Delta \hat{u} + f(u^*) - f(u) + \mathcal{R}_{int}, \quad \forall x \in D \ t \in (0,T), \\ \hat{u}(x,0) &= \mathcal{R}_{tb}(x), \quad \forall x \in D, \\ u(x,t) &= \mathcal{R}_{sb}(x,t), \quad \forall x \in \partial D, \ t \in (0,T). \end{aligned} \tag{3.14}$$

Here, we have denoted $\mathcal{R}_{int} = \mathcal{R}_{int,\theta^*}$ for notational convenience and analogously for the residuals $\mathcal{R}_{tb}, \mathcal{R}_{sb}$.

Multiplying both sides of the PDE (3.14) with $\hat{u}$, integrating over the domain and integrating by parts, denoting $\mathbf{n}$ as the unit outward normal, yields,

$$
\begin{aligned}
\frac{1}{2}\frac{d}{dt}\int_D |\hat{u}(x,t)|^2 dx &= -\int_D |\nabla \hat{u}|^2 dx + \int_{\partial D} \mathcal{R}_{sb}(x,t)(\nabla \hat{u}\cdot \mathbf{n})ds(x) + \int_D \hat{u}(f(u^*)-f(u))dx + \int_D \mathcal{R}_{int}\hat{u}dx. \\
&\leqslant \int_D |\hat{u}||f(u^*)-f(u)|dx + \frac{1}{2}\int_D \hat{u}(x,t)^2 dx + \frac{1}{2}\int_D |\mathcal{R}_{int}|^2 dx \\
&+ \underbrace{|\partial D|^{\frac{1}{2}}\left(\|u\|_{C^1([0,T]\times\partial D)} + \|u^*\|_{C^1([0,T]\times\partial D)}\right)}_{C_{\partial D}(u,u^*)}\left(\int_{\partial D}|\mathcal{R}_{sb}(x,t)|^2 ds(x)\right)^{\frac{1}{2}} \\
&\leqslant (C_f+\frac{1}{2})\int_D |\hat{u}(x,t)|^2 dx + \frac{1}{2}\int_D |\mathcal{R}_{int}|^2 dx + C_{\partial D}(u,u^*)\left(\int_{\partial D}|\mathcal{R}_{sb}(x,t)|^2 ds(x)\right)^{\frac{1}{2}} \quad \text{(by (3.2))}.
\end{aligned}
$$

Integrating the above inequality over $[0,\bar{T}]$ for any $\bar{T}\leqslant T$ and the definition (3.5) together with Cauchy-Schwarz inequality, we obtain,

$$
\begin{aligned}
\int_D |\hat{u}(x,\bar{T})|^2 dx \leqslant \int_D |\mathcal{R}_{tb}(x)|^2 dx + (1+C_f)\int_0^{\bar{T}}\int_D |\hat{u}(x,t)|^2 dxdt + \int_0^T\int_D |\mathcal{R}_{int}|^2 dxdt \\
+ C_{\partial D}(u,u^*)T^{\frac{1}{2}}\left(\int_0^T\int_{\partial D}|\mathcal{R}_{sb}(x,t)|^2 ds(x)dt\right)^{\frac{1}{2}}.
\end{aligned}
$$

Applying the integral form of the Grönwall's inequality to the above, we obtain,

$$
\begin{aligned}
&\int_D |\hat{u}(x,\bar{T})|^2 dx \\
&\leqslant \left(1+(1+2C_f)Te^{(1+2C_f)T}\right)\left(\int_D |\mathcal{R}_{tb}(x)|^2 dx + \int_0^T\int_D |\mathcal{R}_{int}|^2 dxdt + C_{\partial D}(u,u^*)T^{\frac{1}{2}}\left(\int_0^T\int_{\partial D}|\mathcal{R}_{sb}(x,t)|^2 ds(x)dt\right)^{\frac{1}{2}}\right).
\end{aligned}
$$

Integrating over $\bar{T}\in[0,T]$ yields,

$$
\begin{aligned}
&\mathcal{E}_G^2 = \int_0^T\int_D |\hat{u}(x,\bar{T})|^2 dxdt \\
&\leqslant \left(T+(1+2C_f)T^2e^{(1+2C_f)T}\right)\left(\int_D |\mathcal{R}_{tb}(x)|^2 dx + \int_0^T\int_D |\mathcal{R}_{int}|^2 dxdt + C_{\partial D}(u,u^*)T^{\frac{1}{2}}\left(\int_0^T\int_{\partial D}|\mathcal{R}_{sb}(x,t)|^2 ds(x)dt\right)^{\frac{1}{2}}\right).
\end{aligned}
\tag{3.15}
$$

By the definitions of different components of the training error (3.8) and applying the estimates (3.9), (3.10), (3.11) on the quadrature error yields the desired inequality (3.12). □

Remark 3.2. The estimate (3.12) bounds the generalization error in terms of each component of the training error and the quadrature errors. Clearly, each component of the training error can be computed from the loss function (2.16), (3.6), once the training has been completed. As long as the PINN is well-trained i.e, each component of the training error is small, the bound (3.12) implies that the generalization error will be small for large enough number of training points. The estimate is not by any means sharp as triangle inequalities and the Grönwall's inequality are used. Nevertheless, it provides interesting information. For instance, the error due to the boundary residual has a bigger weight in (3.12), relative to the interior and initial residuals. This is consistent with the observations of [23] and can also be seen in the recent papers such as [43] and suggests that the loss function (3.6) could be modified such that the boundary residual is penalized more. ■

Remark 3.3. In addition to the training errors, which could depend on the underlying PDE solution, the estimate (3.12) shows explicit dependence on the underlying solution through the constant $C_{\partial D}$, which is based only on the value of the underlying solution on the boundary. Similarly, the dependence on dimension is only seen through the quadrature error. ■

Remark 3.4. As in subsection 2.4.1, one has to modify the estimate for the generalization error (3.12), when random training points are used. Defining cumulative generalization error $\bar{\mathcal{E}}_G$, analogously to (2.26), by integrating (3.7) over all training sets $\mathbb{S}$, identified with the vector $\mathbf{S}$ and the analogous concepts of cumulative training error $\bar{\mathcal{E}}_T$ (2.27) and validation error $\bar{\mathcal{E}}_V$ (2.29), we can readily combine the arguments of Lemma 2.10 and Theorem 3.1 to obtain the following estimate on the cumulative generalization error,

$$\begin{aligned}\bar{\mathcal{E}}_G^2 &\leqslant C_1^2\left(\left(\bar{\mathcal{E}}_T^{tb}\right)^2+\left(\mathcal{E}_{TV}^{tb}\right)^2+\left(\bar{\mathcal{E}}_T^{int}\right)^2+\left(\mathcal{E}_{TV}^{tb}\right)^2+C_2^2\left(\bar{\mathcal{E}}_T^{sb}+\bar{\mathcal{E}}_{TV}^{sb}\right)\right)\\&+C_1^2\left(\frac{std\left(\mathcal{R}_{tb}^2\right)}{N_{tb}^{\frac{1}{2}}}+\frac{std\left(\mathcal{R}_{int}^2\right)}{N_{int}^{\frac{1}{2}}}+C_2^2\frac{\sqrt{std\left(\mathcal{R}_{sb}^2\right)}}{N_{sb}^{\frac{1}{4}}}\right),\end{aligned}\tag{3.16}$$

with constants $C_{1,2}$ defined in (3.13), standard deviation of the residuals, defined analogously to (2.32) and validation gaps defined by

$$\left(\mathcal{E}_{TV}^{\ell}\right)^2=\left|\left(\bar{\mathcal{E}}_T^{\ell}\right)^2-\left(\bar{\mathcal{E}}_V^{\ell}\right)^2\right|,\quad \ell=sb,tb,int.$$

■

3.4 Numerical experiments.

In this section, we present numerical experiments for the approximation of solutions of the parabolic equation (3.1) by PINNs, generated with algorithm 2.3. We focus on the case of the linear heat equation here, i.e, by setting $f\equiv 0$ in (3.1) as one has explicit formulas for the underlying solution and we use these formulas to explicitly compute the generalization error.

3.4.1 One-dimensional Heat Equation.

For the first numerical experiment, we consider the heat equation in one space dimension i.e, $d=1$ in (3.1) with $f\equiv 0$, domain $D=[-1,1]$, $T=1$, and initial data $\bar{u}(x)=-\sin(\pi x)$. Then, the exact solution of the heat equation is given by,

$$u(x,t)=-\sin(\pi x)e^{-\pi^2 t}.\tag{3.17}$$

Clearly both the initial data and the exact solution are smooth and Theorem 3.1 holds. In particular, we can use any grid based quadrature rule to generate training points in algorithm 2.3. However, to keep the presentation general and allow for very high-dimensional problems later, we simply choose random points for all three components of the training set $\mathbb{S}_{int},\mathbb{S}_{sb},\mathbb{S}_{tb}$. Each set of points is chosen randomly, independently and identically distributed with the underlying uniform distribution. An illustration of these points is provided in figure 2, where the random points in $\mathbb{S}_{int}$ are shown as blue dots, whereas the points in $\mathbb{S}_{sb}$ and $\mathbb{S}_{tb}$ are shown as black crosses. Our first aim in this experiment is to illustrate the estimate (3.16) on the generalization error. To this end, we run algorithm 2.3 with this random training set and with following hyperparameters: we consider a fully connected neural network architecture (2.8), with the tanh activation function, with 4 hidden layers and 20 neurons in each layer, resulting in neural networks with 1761 tuning parameters. Moreover, we use the loss function (2.16), (3.6), with $\lambda=1$ and with $q=2$ i.e L^2-regularization, with regularization parameter $\lambda_{reg}=10^{-6}$. Finally, the optimizer is the second-order LBFGS method. This choice of hyperparameters is consistent with the ensemble training, presented in the next section.

On this hyperparameter configuration, we vary the number of training points as N_{int} = [1000, 2000, 4000, 8000, 16000], $N_{tb}=N_{sb}$ =[8, 16, 32, 64, 256] concurrently, run the algorithm 2.3 to obtain the corresponding trained neural network and evaluate the resulting errors i.e, the cumulative training errors (2.27), and the upper bound in (3.16). The cumulative generalization error (2.26) is computed by evaluating the error of the neural network with respect to the exact solution (3.17) on a *randomly chosen test set* of 10^5 points. We compute the averages and standard deviations by taking $K=30$ different random training sets. The results for this procedure are shown in figure 3. We see from this figure

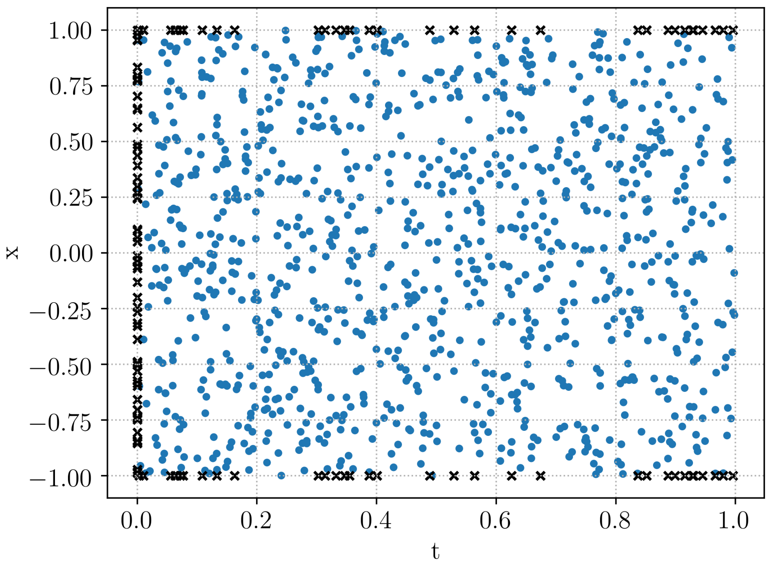


Figure 2: An illustration of the training set $\mathcal{S}$ for the one-dimensional heat equation (3.1) with randomly chosen training points. Points in $\mathcal{S}_{int}$ are depicted with blue dots and those in $\mathcal{S}_{tb} \cup \mathcal{S}_{sb}$ are depicted with black crosses.

that the cumulative generalization error (2.26) is very low to begin with and decays with the number of boundary training points $N_{tb} = N_{sb}$. The effect of the number of interior training points seems to be minimal in this case. Similarly, the computable upper bound (3.16) also decays with respect to increasing the number of boundary training points. However, this upper bound does appear to be a significant overestimate as it is almost three orders to magnitude greater than the actual generalization error. This is not surprising as we had used non-sharp estimates such as triangle inequality and Grönwall's inequality rather indiscriminately while deriving (3.16). Moreover, obtaining sharp bounds on generalization errors is a notoriously hard problem in machine learning [1,35] with overestimates of tens of orders of magnitude. More importantly, both the computed generalization error and the upper bound follow the same decay in the number of training samples. Surprisingly, the training errors (2.27) are slightly larger than the computed generalization errors for this example. Note that this observation is still consistent with the bound (3.16). Given the fact that the training error is defined in terms of residuals and the generalization error is the error in approximating the solution of the underlying PDE by the PINN, there is no reason, a priori, to expect that the generalization error should be greater than the training error.

	$K-1$	d	q	λ_{reg}	λ
1D Heat Equation	2, 4, 8	12, 16, 20	1, 2	0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}	0.01, 0.1, 1, 10
ND Heat Equation	4	20	2	0, 10^{-6}, 10^{-5}	0.1, 1, 10
Burgers Equation	4, 8, 10	16, 20, 24	2	0, 10^{-6}, 10^{-5}	0.1, 1, 10
Euler Equations, Taylor Vortex	4, 8, 12	16, 20, 24	1, 2	0, 10^{-6}, 10^{-5}	0.1, 1, 10
Euler Equations, Double Shear Layer	16, 20, 24	32, 40, 48	1, 2	0, 10^{-6}, 10^{-5}	0.1, 1, 10

Table 1: Hyperparameter configurations employed in the ensemble training of PINNs.

3.4.2 Ensemble training

A PINN involves several hyperparameters, some of which are shown in Table 1. An user is always confronted with the question of which parameter to choose. The theory, presented in this paper and in

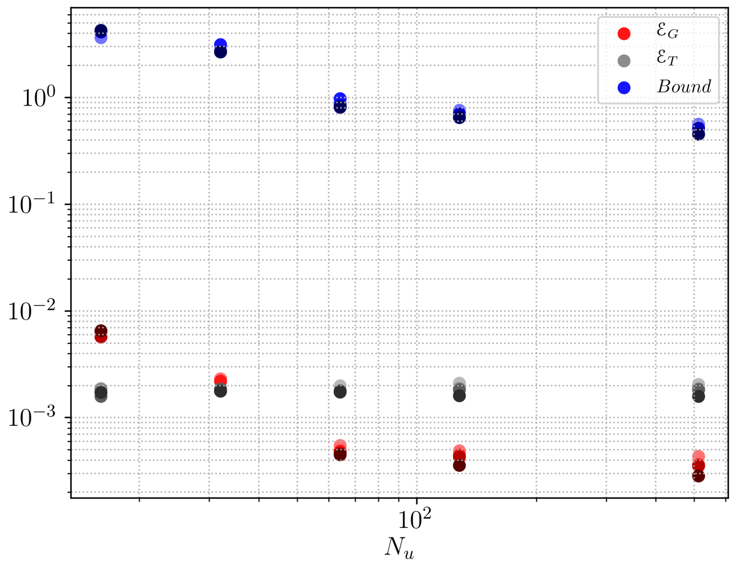


Figure 3: Generalization error, training error and theoretical bound (3.16) VS number of training samples $N_u = N_{sb} + N_{tb}$. Each color gradation corresponds to different values of the number of interior training points N_{int} (from the smallest to the largest).

the literature, offers very little guidance about the choice of hyperparameters. Instead, it is standard practice in machine learning to do a systematic hyperparameter search. To this end, we follow the *ensemble training* procedure of [28] with a randomly chosen training set ($N_{int} = 1024, N_{sb} = N_{tb} = 64$) and compute the marginal distribution of the generalization error with respect to different choices of the hyperparameters (see Table 1). The ensemble training results in a total number of 360 configurations. For each of them, the model is retrained five times with different starting values of the trainable weights in the optimization algorithm and the one resulting in the smallest value of the training loss is selected. We plot the corresponding histograms, visualizing the marginal generalization error distributions, in figure 4.

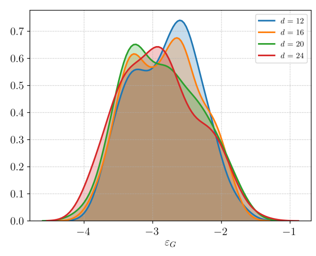
As seen from figure 4, there is a large variation in the spread of the generalization error, often two to three orders of magnitude, indicating sensitivity to hyperparameters. However, even the worst case errors are fairly low for this example. Comparing different hyperparameters, we see that there is not much sensitivity to the network architecture (number of hidden layers and number of neurons per layer) and a slight regularization or no regularization in the loss function (2.16) is preferable to large regularizations. The most sensitive parameter is λ in (3.6) where $\lambda = 1$ or 0.1 are significantly better than larger values of λ. This can be explained in terms of the bounds (3.12), (3.16), where the boundary residual $\mathcal{R}_{sb}$ has a larger weight in error. A smaller value of λ enforces this component of error, and hence the overall error, to be small, and we see exactly this behavior in the results.

Finally, in figure 5, we plot the total training error (3.8) on a logarithmic scale (x-axis) against the generalization error (3.7) (in log scale) (y-axis) for all the hyperparameter configurations in the ensemble training. This plot clearly shows that the two errors are highly correlated and validates the fundamental point of the estimates (3.12) and (3.16) that if the PINNs are trained well, they generalize very well. In other words, low training errors imply low generalization errors.

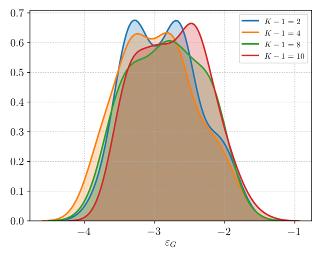
3.4.3 Heat equation in several space dimensions

For this experiment, we consider the linear heat equation in domain $[0,1]^n$ and for the time interval $[0,1]$, for different space dimensions n. We consider the initial data $\bar{u}(x) = \frac{\|x\|^2}{n}$. In this case, the explicit solution of the linear heat equation is given by

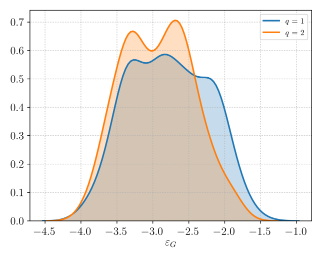
$$u(x,t) = \frac{\|x\|^2}{n} + 2t. \tag{3.18}$$



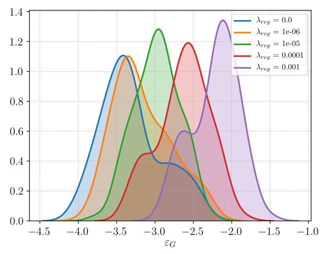
(a) Number of neurons per layer n



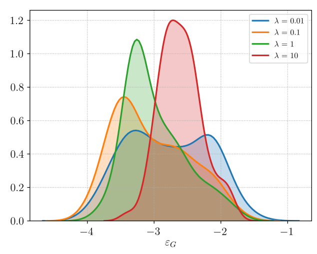
(b) Number of hidden layers $K - 1$



(c) Regularization kernel q



(d) Regularization parameter λ_{reg}



(e) Residual parameter λ

Figure 4: Marginal distributions of the generalization error to different network hyperparameters

For different values of n ranging up to $n = 100$, we generate PINNs with algorithm 2.3, by selecting randomly chosen training points, with respect to the underlying uniform distribution. Ensemble training, as outlined above, is performed in order to select the best performing hyperparameters among the ones listed in Table 1 and resulting errors are shown in 2. In particular, we present the relative percentage (cumulative) generalization error $\mathcal{E}_G^r$, readily computed from definition (3.7) and normalized with the L^2-norm of the exact solution (3.18). We see from the table that the generalization errors are very low, less than 0.1% upto 10 spatial dimensions and rise rather slowly (approximately linearly) with dimension, resulting in a low generalization error of 2.6%, even for 100 space dimensions. This shows the ability of PINNs to possibly overcome the curse of dimensionality, at least for the heat equation. Note that we have not used *any explicit solution formulas* such as the Feynman-Kac formulas in algorithm 2.3. Still, PINNs were able to obtain low enough errors, comparable to supervised learning based neural networks that relied on the availability of an explicit solution formula [3, 11]. This experiment illustrates the ability of PINNs to overcome the *curse of dimensionality*, at least with random training points.

n	N_{int}	N_{sb}	N_{tb}	$K-1$	d	L^1-reg.	L^2-reg.	λ	$\mathcal{E}_T$	$\mathcal{E}_G^r$
1	65536	32768	32768	4	20	0.0	0.0	0.1	$2.8 \cdot 10^{-5}$	0.0035%
5	65536	32768	32768	4	20	0.0	0.0	1.0	0.0002	0.016%
10	65536	32768	32768	4	20	0.0	0.0	0.1	0.0003	0.03%
20	65536	32768	32768	4	20	0.0	10^{-6}	0.1	0.006	0.79%
50	65536	32768	32768	4	20	0.0	10^{-6}	0.1	0.0056	1.5%
100	65536	32768	32768	4	20	0.0	0.0	0.1	0.0035	2.6%

Table 2: Best performing hyperparameters configurations for the multi-dimensional heat, for different values of the dimensions n.

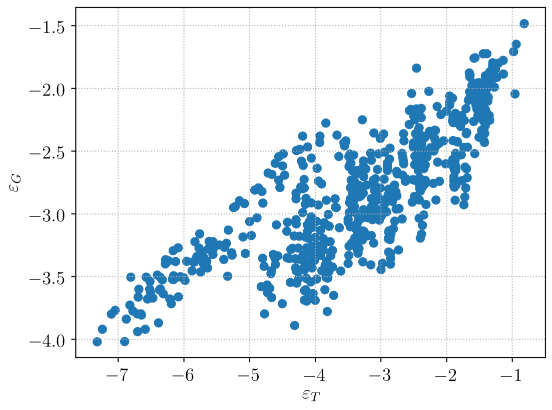


Figure 5: Log of Training error (X-axis) vs Log of Generalization error (Y-axis) for each hyperparameter configuration during ensemble training.

4 Viscous scalar conservation laws

4.1 The underlying PDE

In this section, we consider the following one-dimensional version of *viscous scalar conservation laws* as a model problem for quasilinear, convection-dominated diffusion equations,

$$\begin{aligned} u_t + f(u)_x &= \nu u_{xx}, \quad \forall x \in (0,1),\ t \in [0,T], \\ u(x,0) &= \bar{u}(x), \quad \forall x \in (0,1). \\ u(0,t) &= u(1,t) \equiv 0, \quad \forall t \in [0,T]. \end{aligned} \tag{4.1}$$

Here, $\bar{u} \in C^k([0,1])$, for any $k \geqslant 1$, is the initial data and we consider zero Dirichlet boundary conditions. Note that $0 < \nu \ll 1$ is the viscosity coefficient. The flux function is denoted by $f \in C^k(\mathbb{R};\mathbb{R})$.

We emphasize that (4.1) is a model problem that we present here for notational and expositional simplicity. The following results can be readily extended in the following directions:

- Several space dimensions.
- Other boundary conditions such as Periodic or Neumann boundary conditions.
- More general forms of the viscous term, namely $\nu\left(B(u)u_x\right)_x$, for any $B \in C^k(\mathbb{R};\mathbb{R})$ with $B(v) \geqslant c > 0$, for all $v \in \mathbb{R}$ and for some c.

Moreover, we can follow standard textbooks such as [16] to conclude that as long as $\nu > 0$, there exists a classical solution $u \in C^k([0,T) \times [0,1])$ of the viscous scalar conservation law (4.1).

4.2 PINNs for (4.1)

We realize the abstract algorithm 2.3 in the following concrete steps,

4.2.1 Training Set.

Let $D = (0,1)$ and $D_T = (0,1) \times (0,T)$. As in section 3, we divide the training set $\mathcal{S} = \mathcal{S}_{int} \cup \mathcal{S}_{sb} \cup \mathcal{S}_{tb}$ of the abstract PINNs algorithm 2.3 into the following three subsets,

- Interior training points $\mathcal{S}_{int} = \{y_n\}$ for $1 \leqslant n \leqslant N_{int}$, with each $y_n = (x_n, t_n) \in D_T$. These points can the quadrature points, corresponding to a suitable space -time grid-based composite Gauss quadrature rule or generated from a low-discrepancy sequence in D_T.

- Spatial boundary training points $\mathcal{S}_{sb} = (0,t_n) \cup (1,t_n)$ for $1 \leqslant n \leqslant N_{sb}$, and the points t_n chosen either as Gauss quadrature points or low discrepancy sequences in $[0,T]$.
- Temportal boundary training points $\mathcal{S}_{tb} = \{x_n\}$, with $1 \leqslant n \leqslant N_{tb}$ and each $x_n \in (0,1)$, chosen either as Gauss quadrature points of low-discrepancy sequences.

4.2.2 Residuals

In the algorithm 2.3 for generating PINNs, we need to define appropriate residuals. For the neural network $u_\theta \in C^k([0,T]\times[0,1])$, defined by (2.8), with a smooth activation function such as $\sigma = \tanh$ and $\theta \in \Theta$ as the set of tuning parameters, we define the following residuals,

- Interior Residual given by,

$$\mathcal{R}_{int,\theta}(x,t) := \partial_t(u_\theta(x,t)) + \partial_x(f(u_\theta(x,t))) - \nu\partial_{xx}(u_\theta(x,t)). \tag{4.2}$$

- Spatial boundary Residual given by,

$$\big(\mathcal{R}_{sb,0,\theta}(t),\ \mathcal{R}_{sb,1,\theta}(t)\big) := (u_\theta(0,t),\ u_\theta(1,t)), \quad \forall t \in (0,T]. \tag{4.3}$$

- Temporal boundary Residual given by,

$$\mathcal{R}_{tb,\theta}(x) := u_\theta(x,0) - \bar{u}(x), \quad \forall x \in [0,1]. \tag{4.4}$$

  All the above quantities are well defined for $k \geqslant 2$ and $\mathcal{R}_{int,\theta} \in C^{k-2}([0,1]\times[0,T])$, $\mathcal{R}_{sb,\theta} \in C^k([0,T])$, $\mathcal{R}_{tb,\theta} \in C^k([0,1])$.

4.2.3 Loss function

We use the following loss function to train the PINN for approximating the viscous scalar conservation law (4.1),

$$J(\theta) := \sum_{n=1}^{N_{tb}} w_n^{tb}|\mathcal{R}_{tb,\theta}(x_n)|^2 + \sum_{n=1}^{N_{sb}} w_n^{sb}|\mathcal{R}_{sb,0,\theta}(t_n)|^2 + \sum_{n=1}^{N_{sb}} w_n^{sb}|\mathcal{R}_{sb,1,\theta}(t_n)|^2 + \lambda \sum_{n=1}^{N_{int}} w_n^{int}|\mathcal{R}_{int,\theta}(x_n,t_n)|^2. \tag{4.5}$$

Here the residuals are defined by (4.4), (4.3), (4.2). w_n^{tb} are the N_{tb} quadrature weights corresponding to the temporal boundary training points $\mathcal{S}_{tb}$, w_n^{sb} are the N_{sb} quadrature weights corresponding to the spatial boundary training points $\mathcal{S}_{sb}$ and w_n^{int} are the N_{int} quadrature weights corresponding to the interior training points $\mathcal{S}_{int}$. Furthermore, λ is a hyperparameter for balancing the residuals, on account of the PDE and the initial and boundary data, respectively.

4.3 Estimate on the generalization error.

As for the semilinear parabolic equation, we will try to estimate the following generalization error for the PINN $u^* = u_{\theta^*}$, generated through algorithm 2.3, with loss functions (2.16), (4.5), for approximating the solution of the viscous scalar conservation law (4.1):

$$\mathcal{E}_G := \left(\int_0^T \int_0^1 |u(x,t) - u^*(x,t)|^2 dxdt\right)^{\frac{1}{2}}. \tag{4.6}$$

This generalization error will be estimated in terms of the *training error*,

$$\begin{aligned}\mathcal{E}_T^2 := \lambda \underbrace{\sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{int,\theta^*}(x_n,t_n)|^2}_{(\mathcal{E}_T^{int})^2} + \underbrace{\sum_{n=1}^{N_{tb}} + w_n^{tb} |\mathcal{R}_{tb,\theta^*}(x_n)|^2}_{(\mathcal{E}_T^{tb})^2} \\ + \underbrace{\sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,0,\theta^*}(t_n)|^2}_{(\mathcal{E}_T^{sb,0})^2} + \underbrace{\sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,1,\theta^*}(t_n)|^2}_{(\mathcal{E}_T^{sb,1})^2},\end{aligned} \tag{4.7}$$

readily computed from the training loss (4.5) *a posteriori*. We have the following estimate,

Theorem 4.1. *Let $\nu > 0$ and let $u \in C^k((0,T)\times(0,1))$ be the unique classical solution of the viscous scalar conservation law* (4.1)*. Let $u^* = u_{\theta^*}$ be the PINN, generated by algorithm 2.3, with loss function* (4.5)*. Then, the generalization error* (4.6) *is bounded by,*

$$\begin{aligned}\mathcal{E}_G^2 \leqslant \left(T + \mathbf{C}T^2 e^{\mathbf{C}T}\right)\left[\left(\mathcal{E}_T^{tb}\right)^2 + \left(\mathcal{E}_T^{int}\right)^2 + 2\bar{\mathbf{C}}_b\left(\left(\mathcal{E}_T^{sb,0}\right)^2 + \left(\mathcal{E}_T^{sb,1}\right)^2\right) + 2\nu\mathbf{C}_b T^{\frac{1}{2}}\left(\mathcal{E}_T^{sb,0} + \mathcal{E}_T^{sb,1}\right)\right] \\ + \left(T + \mathbf{C}T^2 e^{\mathbf{C}T}\right)\left[C_{quad}^{tb} N_{tb}^{-\alpha_{tb}} + C_{quad}^{int} N_{int}^{-\alpha_{int}} + 2\bar{\mathbf{C}}_b\left(\left(C_{quad}^{sb,0} + C_{quad}^{sb,1}\right) N_{sb}^{-\alpha_{sb}}\right) + 2\nu\mathbf{C}_b T^{\frac{1}{2}}\left(\left(C_{quad}^{sb,0} + C_{quad}^{sb,1}\right)^{\frac{1}{2}} N_{sb}^{-\frac{\alpha_{sb}}{2}}\right)\right].\end{aligned} \tag{4.8}$$

Here, the training errors are defined by (4.7) *and the constants are given by $\mathbf{C} = 1 + 2C_{f,u,u^*}$, with*

$$\begin{aligned}C_{f,u,u^*} &= C\left(\|f\|_{C^2}, \|u\|_{W^{1,\infty}}, \|u^*\|_{L^\infty}\right) = |f''\left(\max\{\|u\|_{L^\infty}, \|u^*\|_{L^\infty}\}\right)| \, \|u_x\|_{L^\infty}, \\ \mathbf{C}_b &= \left(\|u_x\|_{C([0,1]\times[0,T])} + \|u_x^*\|_{C([0,1]\times[0,T])}\right),\end{aligned} \tag{4.9}$$

$\bar{\mathbf{C}}_b = \bar{\mathbf{C}}_b\left(\|f'\|_\infty, \|u^*\|_{C^0([0,1]\times[0,T])}\right)$ *and* $C_{quad}^{tb} = C_{qaud}^{tb}\left(\|\mathcal{R}_{tb,\theta^*}\|_{C^k}\right)$, $C_{quad}^{int} = C_{qaud}^{int}\left(\|\mathcal{R}_{int,\theta^*}\|_{C^{k-2}}\right)$, $C_{quad}^{sb,0} = C_{qaud}^{sb,0}\left(\|\mathcal{R}_{sb,0,\theta^*}\|_{C^k}\right)$, $C_{quad}^{sb,1} = C_{qaud}^{sb,1}\left(\|\mathcal{R}_{sb,1,\theta^*}\|_{C^k}\right)$ *are constants are appear in the bounds on quadrature error* (3.9)-(3.11).

Proof. We drop the θ^*-dependence of the residuals (4.2)-(4.4) for notational convenience in the following. Define the *entropy flux function*,

$$Q(u) = \int_a^u s f'(s) ds, \tag{4.10}$$

for any $a \in \mathbb{R}$. Let $\hat{u} = u^* - u$ be the error with the PINN. From the PDE (4.1) and the definition of the interior residual (4.2), we have the following identities,

$$\begin{aligned}\partial_t\left(\frac{(u^*)^2}{2}\right) + \partial_x Q(u^*) &= \nu u^* u_{xx}^* + \mathcal{R}_{int} u^* \\ \partial_t\left(\frac{u^2}{2}\right) + \partial_x Q(u) &= \nu u u_{xx}\end{aligned} \tag{4.11}$$

A straightforward calculation with (4.1) and (4.2) yields,

$$\partial_t(u\hat{u}) + \partial_x\left(u\left(f(u^*) - f(u)\right)\right) = \left[f(u^*) - f(u) - f'(u)\hat{u}\right] u_x + \mathcal{R}_{int} u + \nu\left(u\hat{u}_{xx} + \hat{u}u_{xx}\right). \tag{4.12}$$

Subtracting the second equation of (4.11) and (4.12) from the first equation of (4.11) yields,

$$\partial_t S(u,u^*) + \partial_x H(u,u^*) = \mathcal{R}_{int}\hat{u} + T_1 + T_2, \tag{4.13}$$

with,

$$\begin{aligned}S(u,u^*) &:= \frac{(u^*)^2}{2} - \frac{u^2}{2} - \hat{u}u = \frac{1}{2}\hat{u}^2, \\ H(u,u^*) &:= Q(u^*) - Q(u) - u(f(u^*) - f(u)), \\ T_1 &= -\left[f(u^*) - f(u) - f'(u)\hat{u}\right] u_x, \\ T_2 &= \nu\left(u^* u_{xx}^* - u u_{xx} - u\hat{u}_{xx} - \hat{u}u_{xx}\right) = \nu\hat{u}\hat{u}_{xx}.\end{aligned}$$

As the flux f is smooth, by a Taylor expansion, we see that

$$T_1 = -f''(u + \gamma(u^* - u))\hat{u}^2 u_x, \tag{4.14}$$

for some $\gamma \in (0,1)$. Hence, a straightforward estimate for T_1 is given by,

$$|T_1| \leqslant C_{f,u,u^*}\hat{u}^2, \tag{4.15}$$

with C_{f,u,u^*} defined in (4.9). Next, we integrate (4.13) over the domain $(0,1)$ and integrate by parts to obtain,

$$\begin{aligned}\frac{d}{dt}\int_0^1 \hat{u}^2(x,t)dx &\leqslant 2H(u(0,t),u^*(0,t)) - 2H(u(1,t),u^*(1,t)) \\ &+ \mathbf{C}\int_0^1 \hat{u}^2(x,t)dx + \int_0^1 \mathcal{R}_{int}^2(x,t)dx, \\ &- 2\nu\int_0^1 \hat{u}_x^2(x,t)dx + 2\nu\left(\hat{u}(1,t)\hat{u}_x(1,t) - \hat{u}(0,t)\hat{u}_x(0,t)\right),\end{aligned} \tag{4.16}$$

with the constant, $\mathbf{C} = 1 + 2C_{f,u,u^*}$.

Next, for any $\bar{T} \leqslant T$, we estimate the boundary terms starting with,

$$\begin{aligned}\int_0^{\bar{T}} \hat{u}(0,t)\hat{u}_x(0,t)dt &= \int_0^{\bar{T}} \mathcal{R}_{sb,0}(t)\left(u_x^*(0,t) - u_x(0,t)\right)dt \\ &\leqslant \underbrace{\left(\|u_x\|_{C([0,1]\times[0,T])} + \|u_x^*\|_{C([0,1]\times[0,T])}\right)}_{\mathbf{C}_b} T^{\frac{1}{2}}\left(\int_0^T \mathcal{R}_{sb,0}^2(t)dt\right)^{\frac{1}{2}}.\end{aligned}$$

Analogously we can estimate,

$$\int_0^{\bar{T}} \hat{u}(1,t)\hat{u}_x(1,t)dt \leqslant \mathbf{C}_b T^{\frac{1}{2}}\left(\int_0^T \mathcal{R}_{sb,1}^2(t)dt\right)^{\frac{1}{2}}.$$

We can also estimate from (4.1) and (3.4) that,

$$\begin{aligned}H(u(0,t),u^*(0,t)) &= Q(u^*(0,t)) - Q(u(0,t)) - u(0,t)(f(u^*(0,t)) - f(u(0,t))), \\ &= Q(\mathcal{R}_{sb,0}(t)) - Q(0), \quad \text{as } u(0,t) = 0, \\ &= Q'(\gamma_0\mathcal{R}_{sb,0}(t))\mathcal{R}_{sb,0}(t), \quad \text{for some } \gamma_0 \in (0,1), \\ &= \gamma_0 f'(\gamma_0 u^*(0,t))\mathcal{R}_{sb,0}^2(t), \quad \text{by (4.10)}, \\ &\leqslant \bar{\mathbf{C}}_b\mathcal{R}_{sb,0}^2(t), \quad \text{with } \bar{\mathbf{C}}_b = \bar{\mathbf{C}}_b\left(\|f'\|_\infty, \|u^*\|_{C^0([0,1]\times[0,T])}\right).\end{aligned}$$

Analogously, we can estimate,

$$H(u(1,t),u^*(1,t)) \leqslant \bar{\mathbf{C}}_b\mathcal{R}_{sb,1}^2(t).$$

For any $\bar{T} \leqslant T$, integrating (4.16) over the time interval $[0,\bar{T}]$ and using the above inequalities on the boundary terms, together with the definition of the residual (4.4) yields,

$$\begin{aligned}\int_0^1 \hat{u}^2(x,\bar{T})dx &\leqslant \mathcal{C} + \mathbf{C}\int_0^{\bar{T}}\int_0^1 \hat{u}^2(x,t)dxdt, \\ \mathcal{C} &= \int_0^1 \mathcal{R}_{tb}^2(x)dx + \int_0^T\int_0^1 \mathcal{R}_{int}^2(x,t)dxdt \\ &+ 2\bar{\mathbf{C}}_b\left[\int_0^T \mathcal{R}_{sb,0}^2(t)dt + \int_0^T \mathcal{R}_{sb,1}^2(t)dt\right] + 2\nu\mathbf{C}_b T^{\frac{1}{2}}\left[\left(\int_0^T \mathcal{R}_{sb,0}^2(t)dt\right)^{\frac{1}{2}} + \left(\int_0^T \mathcal{R}_{sb,1}^2(t)dt\right)^{\frac{1}{2}}\right].\end{aligned} \tag{4.17}$$

By applying the integral form of the Grönwall's inequality to (4.17) for any $\bar{T} \leqslant T$ and integrating again over $\bar{T}$, together with the definition of the generalization error (4.6), we obtain,

$$\mathcal{E}_G^2 \leqslant \left(T + \mathbf{C}T^2 e^{\mathbf{C}T}\right) \mathcal{C}. \tag{4.18}$$

Using the bounds (3.9)-(3.11) on the quadrature errors and the definition of $\mathcal{C}$ in (4.17), we obtain,

$$\begin{aligned}
\mathcal{C} \leqslant & \sum_{n=1}^{N_{tb}} w_n^{tb} |\mathcal{R}_{tb}(x_n)|^2 + C_{qaud}^{tb} \left(\|\mathcal{R}_{tb}\|_{C^k}\right) N_{tb}^{-\alpha_{tb}} \\
& + \sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{int}(x_n,t_n)|^2 + C_{qaud}^{int} \left(\|\mathcal{R}_{int}\|_{C^{k-2}}\right) N_{int}^{-\alpha_{int}}, \\
& + 2\bar{\mathbf{C}}_b \left[\sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,0}(t_n)|^2 + \sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,1}(t_n)|^2 + \left(C_{qaud}^{sb}\left(\|\mathcal{R}_{sb,0}\|_{C^k}\right) + C_{qaud}^{sb}\left(\|\mathcal{R}_{sb,1}\|_{C^k}\right)\right) N_{sb}^{-\alpha_{sb}}\right] \\
& + 2\nu \mathbf{C}_b T^{\frac{1}{2}} \left[\left(\sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,0}(t_n)|^2\right)^{\frac{1}{2}} + \left(\sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,1}(t_n)|^2\right)^{\frac{1}{2}} + \left(C_{qaud}^{sb}\left(\|\mathcal{R}_{sb,0}\|_{C^k}\right) + C_{qaud}^{sb}\left(\|\mathcal{R}_{sb,1}\|_{C^k}\right)\right)^{\frac{1}{2}} N_{sb}^{-\frac{\alpha_{sb}}{2}}\right].
\end{aligned}$$

From definition of training errors (4.7) and (4.18) and the above inequality, we obtain the desired estimate (4.8). □

Remark 4.2. The estimate (4.8) is a concrete realization of the abstract estimate (2.21), with training error decomposed into 4 parts, the constants, associated with the PDE, are given by $C_{f,u,u^*}, \mathbf{C}_b$ (4.9) and the constants due to the quadrature errors are also clearly delineated. ■

Remark 4.3. A close inspection of the estimate (4.8) reveals that at the very least, the classical solution u of the PDE (4.1) needs to be in $L^\infty((0,T); W^{1,\infty}((0,1)))$ for the rhs of (4.8) to be bounded. This indeed holds as long as $\nu > 0$. However, it is well known (see [16] and references therein) that if u^ν is the solution of (4.1) for viscosity ν, then for some initial data,

$$\|u^\nu\|_{L^\infty((0,T);W^{1,\infty}((0,1)))} \sim \frac{1}{\sqrt{\nu}}. \tag{4.19}$$

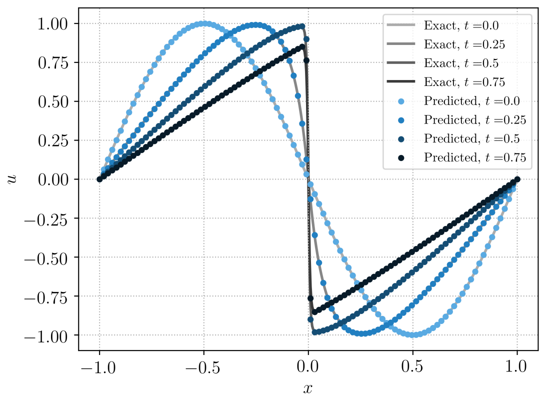
Thus, in the limit $\nu \longrightarrow 0$, the constant C_{f,u,u^*} can blow up (exponentially in time) and the bound (4.8) no longer controls the generalization error. This is not unexpected as the whole strategy of this paper relies on pointwise realization of residuals. However, the zero-viscosity limit of (4.1), leads to a scalar conservation law with discontinuous solutions (shocks) and the residuals are measures that do not make sense pointwise. Thus, the estimate (4.8) also points out the limitations of a PINN for approximating discontinuous solutions. ■

4.4 Numerical experiments

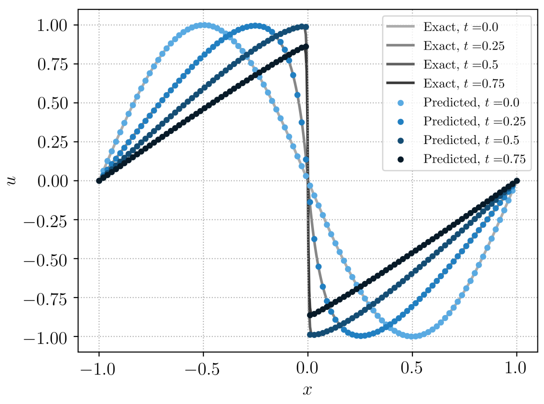
We consider the viscous scalar conservation law (4.1), but in the domain $D_T = [-1,1] \times [0,1]$, with initial conditions, $\bar{u}(x) = -\sin(\pi x)$ and zero Dirichlet boundary conditions. We choose the flux function $f(u) = \frac{u^2}{2}$, resulting in the well-known *viscous Burgers'* equation.

This problem is considered for 4 different values of the viscosity parameter $\nu = \frac{c}{\pi}$, with $c = 0.01$, 0.005, 0.001, 0.0, and with $N_{int} = 8192$, $N_{tb} = 256$, $N_{sb} = 256$ points. All the training points are chosen as low-discrepancy Sobol sequences on the underlying domains. An ensemble training procedure, based on the hyperparameters presented in Table 1, is performed and the best performing hyperparameters i.e. those that led to the smallest training errors, are chosen and presented in Table 3.

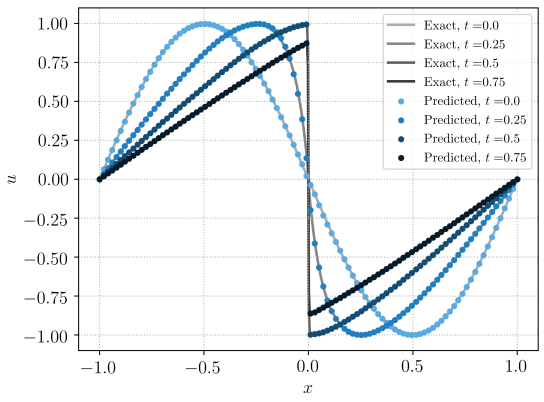
In figure 6, we present the *reference* solution field $u(\cdot,t)$, at different time snapshots, of the viscous Burgers' equation computed with a simple upwind finite volume scheme and forward Euler time integration with 2×10^6 Cartesian grid points in space-time, and the predicted solution $u^*(\cdot,t)$ of the PINN, generated with algorithm 2.3, corresponding to the best performing hyperparameters (see Table 3), for different values



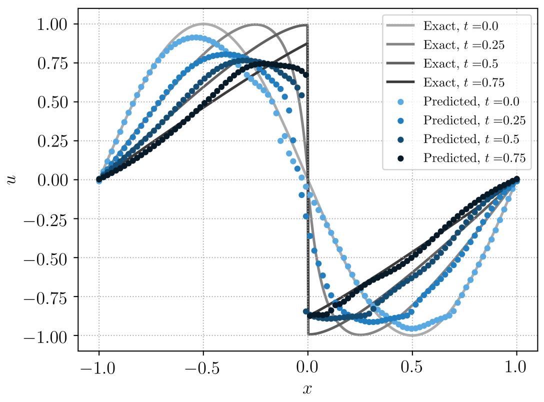
(a) $\nu = \frac{0.01}{\pi}$, $\mathcal{E}_G^r = 0.010$



(b) $\nu = \frac{0.005}{\pi}$, $\mathcal{E}_G^r = 0.012$



(c) $\nu = \frac{0.001}{\pi}$, $\mathcal{E}_G^r = 0.11$



(d) $\nu = 0$, $\mathcal{E}_G^r = 0.23$

Figure 6: Burgers equation with discontinuos solution for different values of ν

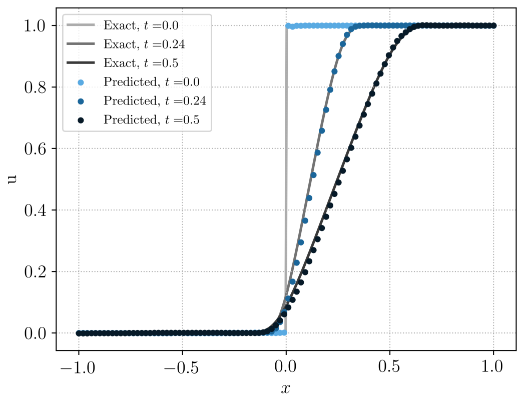
of the viscosity coefficient. From this figure, we observe that for the viscosity coefficients, corresponding to $c = 0.01, 0.005$, the approximate solution, predicted by the PINN, approximates the underlying exact solution, which involves self steepening of the initial sine wave into a steady sharp profile at the origin, very well. This is further reinforced by the very low (relative percentage) generalization errors of approximately 1%, presented in Table 3. However, this efficient approximation is no longer the case for the inviscid problem i.e $\nu = 0$. As seen from figure 6 (bottom right), the PINN fails to resolve the solution, which in this case, consists of a steady shock at the origin. In fact, this failure to approximate is already seen with a viscosity coefficient of $\nu = \frac{0.001}{\pi}$. For this very low viscosity coefficient, we see that the relative generalization error is approximately 11%. The generalization error rises to 23% for the inviscid Burgers' equation. This increase in error appears consistent with the bound (4.8), combined with the blow-up estimate (4.19) for the derivatives of the viscous Burgers' equation. As the viscosity $\nu \to 0$, the bounds in the rhs of (4.8) can increase exponentially, which appears to be the case here.

To further test the bound (4.8)'s ability to explain performance of PINNs for the viscous Burgers' equation, we consider the following initial and boundary conditions,

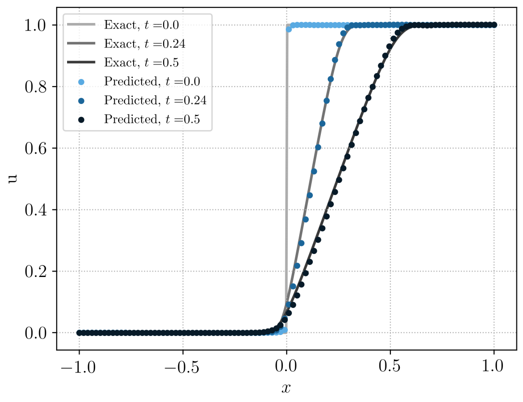
$$\bar{u}(x) = \begin{cases} 0, & \text{if } x \leqslant 0 \\ 1, & \text{if } x > 0 \end{cases} \quad \forall\ x \in [-1, 1],$$

$$u(t, -1) = 0,\ u(t, 1) = 1,\ \forall\ t \in [0, 0.5]. \tag{4.20}$$

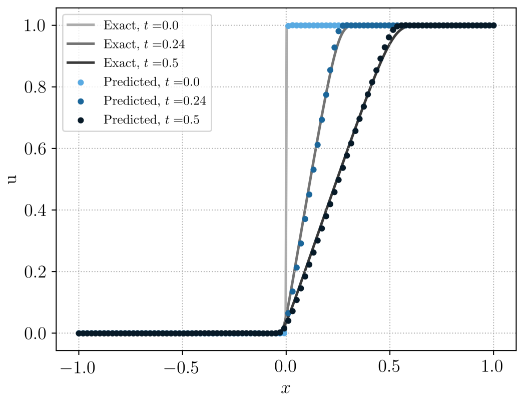
Given the discontinuity in the initial data we train the PINNS with a larger number of boundary training samples $N_{tb} = 512$ and $N_{sb} = 512$, while leaving $N_{int} = 8192$, unchanged. As in the previous



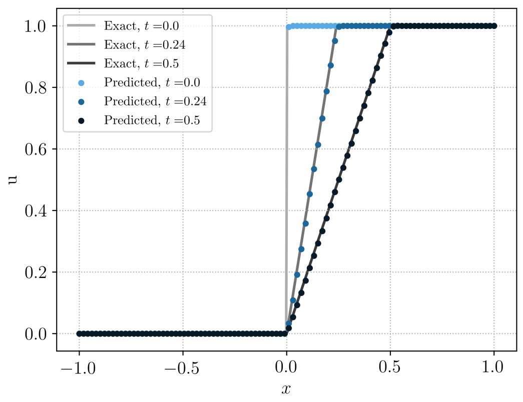
(a) $\nu = \frac{0.01}{\pi}$, $\mathcal{E}_G^r = 0.022$



(b) $\nu = \frac{0.005}{\pi}$, $\mathcal{E}_G^r = 0.018$



(c) $\nu = \frac{0.001}{\pi}$, $\mathcal{E}_G^r = 0.016$



(d) $\nu = 0$, $\mathcal{E}_G^r = 0.012$

Figure 7: Burgers equation with rarefaction wave for different values of ν

experiments, training sets are Sobol sequences and an ensemble training is preformed to configure the network architecture. The results are summarized in Table 4 and figure 7. In this case, the exact solution is a so-called rarefaction wave (see figure 7 for the reference solution, computed in the manner, analogous to the previous numerical experiment) and the gradient of the solution remains bounded, uniformly as the viscosity coefficient $\nu \to 0$. Hence, from the bound (4.8), we expect that PINNs will efficiently approximate the underlying solution for all values of the viscosity coefficient. This is indeed verified in the solution snapshots, presented figure 7, where we observe that the PINN approximates the reference solution quite well, for all values of the viscosity coefficient. This behavior is further verified in table 4, where we see that the generalization error (4.6), remains low (less than 2%) for all the values of viscosity and in fact, reduces slightly as $\nu \to 0$, completely validating the error estimate (4.8).

ν	N_{int}	N_{sb}	N_{tb}	$K-1$	d	L^1-reg.	L^2-reg.	λ	$\mathcal{E}_T$	$\mathcal{E}_G^r$
$0.01/pi$	8192	256	256	8	20	0.0	0.0	0.1	0.0005	1.0%
$0.005/pi$	8192	256	256	10	20	0.0	0.0	0.1	0.00075	1.2%
$0.001/pi$	8192	256	256	10	20	0.0	10^{-6}	0.1	0.009	11.0%
0.0	8192	256	256	8	24	0.0	10^{-5}	0.1	0.08	23.0%

Table 3: Best performing *Neural Network* configurations for the Burgers equation with shock, for different values of the parameter ν.

ν	N_{int}	N_{sb}	N_{tb}	$K-1$	d	L^1-reg.	L^2-reg.	λ	$\mathcal{E}_T$	$\mathcal{E}_G^r$
$0.01/pi$	8192	512	512	4	20	0.0	0.0	0.1	0.0043	2.2%
$0.005/pi$	8192	512	512	4	20	0.0	0.0	0.1	0.0034	1.8%
$0.001/pi$	8192	512	512	4	16	0.0	0.0	0.1	0.00048	1.6%
0.0	8192	512	512	4	20	0.0	0.0	0.1	0.00033	1.2%

Table 4: Best performing *Neural Network* configurations for the Burgers equation with rarefaction wave, for different values of the parameter ν.

5 Incompressible Euler equations

5.1 The underlying PDE

The motion of an inviscid, incompressible fluid is modeled by the incompressible Euler equations [30]. We consider the following form of these PDEs,

$$\begin{aligned}
\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\,\mathbf{u} + \nabla p &= \mathbf{f}, \quad (x,t) \in D \times (0,T), \\
\mathrm{div}(\mathbf{u}) &= 0, \quad (x,t) \in D \times (0,T), \\
\mathbf{u}\cdot\mathbf{n} &= 0, \quad (x,t) \in \partial D \times (0,T), \\
\mathbf{u}(x,0) &= \bar{\mathbf{u}}(x), \quad x \in D.
\end{aligned} \tag{5.1}$$

Here, $D \subset \mathbb{R}^d$, for $d = 2,3$ is an open, bounded, connected subset with smooth C^1 boundary ∂D, $D_T = D \times (0,T)$, $\mathbf{u} : D_T \mapsto \mathbb{R}^d$ is the velocity field, $p : D_T \mapsto \mathbb{R}$ is the pressure that acts as a Lagrange multiplier to enforce the divergence constraint and $\mathbf{f} \in C^1(D_T;\mathbb{R}^d)$ is a forcing term. We use the *no penetration* boundary conditions here with $\mathbf{n}$ denoting the unit outward normal to ∂D.

Note that we have chosen to present this form of the incompressible Euler equations for simplicity of exposition. The analysis, presented below, can be readily but tediously extended to the following,

- Other boundary conditions such as periodic boundary conditions on the torus $\mathbb{T}^d$.
- The *Navier-Stokes equations*, where we add the *viscous term* $\nu\Delta\mathbf{u}$ to the first equation in (5.1), with either periodic boundary conditions or the so-called *no slip* boundary conditions i.e, $\mathbf{u} \equiv 0$, for all $x \in \partial D$ and for all $t \in (0,T]$.

5.2 PINNs

We describe the algorithm 2.3 for this PDE in the following steps,

5.2.1 Training set

We chose the training set $\mathcal{S} \subset D_T$ with $\mathcal{S} = \mathcal{S}_{int} \cup \mathcal{S}_{sb} \cup \mathcal{S}_{tb}$, with interior, spatial and temporal boundary training sets, chosen exactly as in section 3.2.1, either as quadrature points for a (composite) Gauss rule or as low-discrepancy sequences on the underlying domains.

5.2.2 Residuals

For the neural networks $(x,t) \mapsto (\mathbf{u}_\theta(x,t), p_\theta(x,t)) \in C^k((0,T)\times D) \cap C([0,T]\times \bar{D})$, defined by (2.8), with a smooth activation function and $\theta \in \Theta$ as the set of tuning parameters, we define the residual $\mathcal{R}$ in algorithm 2.3, consisting of the following parts,

- *Velocity residual* given by,

$$\mathcal{R}_{\mathbf{u},\theta}(x,t) := (\mathbf{u}_\theta)_t + (\mathbf{u}_\theta \cdot \nabla)\,\mathbf{u}_\theta + \nabla p_\theta - \mathbf{f}, \quad (x,t) \in D \times (0,T), \tag{5.2}$$

- *Divergence residual* given by,

$$\mathcal{R}_{div,\theta}(x,t) := \mathrm{div}(\mathbf{u}_\theta(x,t)), \quad (x,t) \in D \times (0,T), \tag{5.3}$$

- *Spatial boundary Residual* given by,

$$\mathcal{R}_{sb,\theta}(x,t) := \mathbf{u}_\theta(x,t)\cdot \mathbf{n}, \quad \forall x \in \partial D,\ t \in (0,T]. \tag{5.4}$$

- *Temporal boundary Residual* given by,

$$\mathcal{R}_{tb,\theta}(x) := \mathbf{u}_\theta(x,0) - \bar{\mathbf{u}}(x), \quad \forall x \in D. \tag{5.5}$$

As the underlying neural networks have the required regularity, the residuals are well-defined.

5.2.3 Loss function

We consider the following loss function for training PINNs to approximate the incompressible Euler equation (5.1),

$$J(\theta) := \sum_{n=1}^{N_{tb}} w_n^{tb} |\mathcal{R}_{tb,\theta}(x_n)|^2 + \sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,\theta}(x_n,t_n)|^2 + \lambda \left(\sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{\mathbf{u},\theta}(x_n,t_n)|^2 + \sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{div,\theta}(x_n,t_n)|^2 \right). \tag{5.6}$$

Here the residuals are defined by (5.2)-(5.5). w_n^{tb} are the N_{tb} quadrature weights corresponding to the temporal boundary training points $\mathcal{S}_{tb}$, w_n^{sb} are the N_{sb} quadrature weights corresponding to the spatial boundary training points $\mathcal{S}_{sb}$ and w_n^{int} are the N_{int} quadrature weights corresponding to the interior training points $\mathcal{S}_{int}$. Furthermore, λ is a hyperparameter for balancing the residuals, on account of the PDE and the initial and boundary data, respectively.

5.3 Estimate on the generalization error.

We denote the PINN, obtained by the algorithm 2.3, for approximating the incompressible Euler equations, as $\mathbf{u}^* = \mathbf{u}_{\theta^*}$, with θ^* being a (approximate,local) minimum of the loss function (2.16),(5.6). We consider the following generalization error,

$$\mathcal{E}_G := \left(\int_0^T \int_D \|\mathbf{u}(x,t) - \mathbf{u}^*(x,t)\|^2 dx dt \right)^{\frac{1}{2}}, \tag{5.7}$$

with $\|\cdot\|$ denoting the Euclidean norm in $\mathbb{R}^d$. Note that we only consider the error with respect to the velocity field $\mathbf{u}$ in (5.7). Although the pressure p in (5.1) is approximated by the neural network $p^* = p_{\theta^*}$, we recall that the pressure is a Lagrange multiplier, and not a primary variable in the incompressible Euler equations. Hence, we will not consider pressure errors here.

As in section 2, we will bound the generalization error in terms of the following *training errors*,

$$\mathcal{E}_T^2 := \underbrace{\sum_{n=1}^{N_{tb}} w_n^{tb} |\mathcal{R}_{tb,\theta^*}(x_n)|^2}_{(\mathcal{E}_T^{tb})^2} + \underbrace{\sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb,\theta^*}(x_n,t_n)|^2}_{(\mathcal{E}_T^{sb})^2} + \underbrace{\sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{\mathbf{u},\theta^*}(x_n,t_n)|^2}_{(\mathcal{E}_T^{\mathbf{u}})^2} + \lambda \underbrace{\sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{div,\theta^*}(x_n,t_n)|^2}_{(\mathcal{E}_T^{d})^2}. \tag{5.8}$$

As in the previous sections, the training errors can be readily computed *a posteriori* from the loss function (2.16), (5.6).

We have the following bound on the generalization error in terms of the training error,

Theorem 5.1. *Let* $\mathbf{u} \in C^1((0,T)\times D) \cap C([0,T]\times \bar{D})$ *be the classical solution of the incompressible Euler equations* (5.1). *Let* $\mathbf{u}^* = \mathbf{u}_{\theta^*}, p^* = p_{\theta^*}$ *be the PINN generated by algorithm 2.3, then the resulting generalization error* (5.7) *is bounded as,*

$$\begin{aligned}\mathcal{E}_G^2 \leqslant &\left(T + C_\infty T^2 e^{C_\infty T}\right)\left[\left(\mathcal{E}_T^{tb}\right)^2 + \left(\mathcal{E}_T^{\mathbf{u}}\right)^2 + C_0 T^{\frac{1}{2}}\left(\mathcal{E}_T^{div} + \mathcal{E}_T^{sb}\right)\right]\\ &+ \left(T + C_\infty T^2 e^{C_\infty T}\right)\left[C_{qaud}^{tb} N_{tb}^{-\alpha_{tb}} + C_{qaud}^{int,\mathbf{u}} N_{int}^{-\alpha_{int}} + \left(C_{qaud}^{int,div}\right)^{\frac{1}{2}} N_{int}^{-\frac{\alpha_{int}}{2}} + \left(C_{qaud}^{sb}\right)^{\frac{1}{2}} N_{sb}^{-\frac{\alpha_{sb}}{2}}\right].\end{aligned} \tag{5.9}$$

Here, the training errors are defined in (5.8) *and the constants are given by,*

$$\begin{aligned} C_0 &= C\left(\|\mathbf{u}\|_{C^0([0,T]\times\bar{D})}, \|\mathbf{u}^*\|_{C^0([0,T]\times\bar{D})}, \|p\|_{C^0([0,T]\times\bar{D})}, \|p^*\|_{C^0([0,T]\times\bar{D})}\right),\\ C_\infty &= 1 + 2C_d\|\nabla \mathbf{u}\|_{L^\infty(D_T)},\end{aligned} \tag{5.10}$$

with C_d *only depending on dimension* d *and* $C_{quad}^{tb} = C_{qaud}^{tb}\left(\|\mathcal{R}_{tb,\theta^*}\|_{C^k}\right)$, $C_{quad}^{int,\mathbf{u}} = C_{qaud}^{int}\left(\|\mathcal{R}_{\mathbf{u},\theta^*}\|_{C^{k-1}}\right)$, $C_{quad}^{int,div} = C_{qaud}^{int}\left(\|\mathcal{R}_{div,\theta^*}\|_{C^{k-1}}\right)$ *and* $C_{quad}^{sb} = C_{quad}^{sb}\left(\|\mathcal{R}_{sb,\theta^*}\|_{C^k}\right)$ *are the constants associated with the quadrature errors* (3.9)-(3.11).

Proof. We will drop explicit dependence of all quantities on the parameters θ^* for notational convenience. We denote the difference between the underlying solution $\mathbf{u}$ of (5.1) and PINN $\mathbf{u}^*$ as $\hat{\mathbf{u}} = \mathbf{u}^* - \mathbf{u}$. Similarly $\hat{p} = p^* - p$. Using the PDE (5.1) and the definitions of the residuals (5.2)-(5.5), a straightforward calculation yields the following PDE for the $\hat{\mathbf{u}}$,

$$\begin{aligned} \hat{\mathbf{u}}_t + (\hat{\mathbf{u}}\cdot\nabla)\hat{\mathbf{u}} + (\mathbf{u}\cdot\nabla)\hat{\mathbf{u}} + (\hat{\mathbf{u}}\cdot\nabla)\mathbf{u} + \nabla\hat{p} &= \mathcal{R}_{\mathbf{u}}, \quad (x,t)\in D\times(0,T),\\ \mathrm{div}(\hat{\mathbf{u}}) &= \mathcal{R}_{div}, \quad (x,t)\in D\times(0,T),\\ \hat{\mathbf{u}}\cdot\mathbf{n} &= \mathcal{R}_{sb}, \quad (x,t)\in\partial D\times(0,T),\\ \mathbf{u}(x,0) &= \mathcal{R}_{tb}, \quad x\in D.\end{aligned} \tag{5.11}$$

We take a inner product of the first equation in (5.11) with the vector $\hat{\mathbf{u}}$ and use the following vector identities,

$$\begin{aligned} \hat{\mathbf{u}}\cdot\partial_t\hat{\mathbf{u}} &= \partial_t\left(\frac{\|\hat{\mathbf{u}}\|^2}{2}\right),\\ \hat{\mathbf{u}}\cdot((\hat{\mathbf{u}}\cdot\nabla)\hat{\mathbf{u}}) &= (\hat{\mathbf{u}}\cdot\nabla)\left(\frac{\|\hat{\mathbf{u}}\|^2}{2}\right),\\ \hat{\mathbf{u}}\cdot((\mathbf{u}\cdot\nabla)\hat{\mathbf{u}}) &= (\mathbf{u}\cdot\nabla)\left(\frac{\|\hat{\mathbf{u}}\|^2}{2}\right),\end{aligned}$$

yields the following identity,

$$\partial_t\left(\frac{\|\hat{\mathbf{u}}\|^2}{2}\right) + (\hat{u}\cdot\nabla)\left(\frac{\|\hat{\mathbf{u}}\|^2}{2}\right) + (\mathbf{u}\cdot\nabla)\left(\frac{\|\hat{\mathbf{u}}\|^2}{2}\right) + \hat{\mathbf{u}}\cdot((\hat{\mathbf{u}}\cdot\nabla)\mathbf{u}) + (\hat{\mathbf{u}}\cdot\nabla)\hat{p} = \hat{\mathbf{u}}\cdot\mathcal{R}_{\mathbf{u}}.$$

Integrating the above identity over D and integrating by parts, together with (5.1) and (5.11) yields,

$$\begin{aligned}\frac{d}{dt}\int_D\left(\frac{\|\hat{\mathbf{u}}\|^2}{2}\right)dx &= \int_D \mathcal{R}_{div}\left(\frac{\|\hat{\mathbf{u}}\|^2}{2} + \hat{p}\right)dx - \int_{\partial D}\mathcal{R}_{sb}\left(\frac{\|\hat{\mathbf{u}}\|^2}{2} + \hat{p}\right)ds(x)\\ &\quad - \int_D \hat{\mathbf{u}}\cdot((\hat{\mathbf{u}}\cdot\nabla)\mathbf{u})\,dx + \int_D \hat{\mathbf{u}}\cdot\mathcal{R}_{\mathbf{u}}dx.\end{aligned} \tag{5.12}$$

It is straightforward to obtain the following inequality,

$$\int_D \hat{\mathbf{u}} \cdot ((\hat{\mathbf{u}} \cdot \nabla)\, \mathbf{u})\, dx \leqslant C_d \|\nabla \mathbf{u}\|_\infty \int_D \|\hat{\mathbf{u}}\|^2 dx,$$

with the constant C_d only depending on dimension and $\|\mathbf{u}\|_\infty = \|\mathbf{u}\|_{L^\infty(D_T)}$.

Using the above estimate and estimating (5.12) yields,

$$\frac{d}{dt}\int_D \|\hat{\mathbf{u}}\|^2 dx \leqslant C_0 \left[\left(\int_D (\mathcal{R}_{div})^2\, dx\right)^{\frac{1}{2}} + \left(\int_{\partial D} (\mathcal{R}_{sb})^2\, ds(x)\right)^{\frac{1}{2}}\right] + C_\infty \int_D \|\hat{\mathbf{u}}\|^2 dx + \int_D \mathcal{R}_{\mathbf{u}}^2 dx, \tag{5.13}$$

with constants given by (5.10).

For any $\bar{T} \leqslant T$, we integrate (5.13) over time and use some simple inequalities to obtain,

$$\begin{aligned}
\int_D \|\hat{\mathbf{u}}(x,\bar{T})\|^2 dx &\leqslant \mathcal{C} + C_\infty \int_0^{\bar{T}} \int_D \|\hat{\mathbf{u}}(x,t)\|^2 dx dt, \\
\mathcal{C} &= \int_D \mathcal{R}_{tb}^2 dx + \int_0^T \int_D \mathcal{R}_{\mathbf{u}}^2 dx dt, \\
&+ C_0 T^{\frac{1}{2}} \left[\left(\int_0^T \int_D (\mathcal{R}_{div})^2\, dx dt\right)^{\frac{1}{2}} + \left(\int_0^T \int_{\partial D} (\mathcal{R}_{sb})^2\, ds(x) dt\right)^{\frac{1}{2}}\right].
\end{aligned} \tag{5.14}$$

Now by using the integral form of the Grönwall's inequality in (5.14) and integrating again over $[0,T]$ results in,

$$\mathcal{E}_G^2 \leqslant \left(T + C_\infty T^2 e^{C_\infty T}\right) \mathcal{C}. \tag{5.15}$$

Using the bounds (3.9)-(3.11) on the quadrature errors and the definition of $\mathcal{C}$ in (5.14), we obtain,

$$\begin{aligned}
\mathcal{C} &\leqslant \sum_{n=1}^{N_{tb}} w_n^{tb} |\mathcal{R}_{tb}(x_n)|^2 + C_{qaud}^{tb} \left(\|\mathcal{R}_{tb}\|_{C^k}\right) N_{tb}^{-\alpha_{tb}} \\
&+ \sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{\mathbf{u}}(x_n,t_n)|^2 + C_{qaud}^{int} \left(\|\mathcal{R}_{\mathbf{u}}\|_{C^{k-1}}\right) N_{int}^{-\alpha_{int}}, \\
&+ C_0 T^{\frac{1}{2}} \left[\left(\sum_{n=1}^{N_{int}} w_n^{int} |\mathcal{R}_{div}(x_n,t_n)|^2\right)^{\frac{1}{2}} + \left(C_{qaud}^{int} \left(\|\mathcal{R}_{div}\|_{C^{k-1}}\right)\right)^{\frac{1}{2}} N_{int}^{-\frac{\alpha_{int}}{2}}\right] \\
&+ C_0 T^{\frac{1}{2}} \left[\left(\sum_{n=1}^{N_{sb}} w_n^{sb} |\mathcal{R}_{sb}(x_n,t_n)|^2\right)^{\frac{1}{2}} + \left(C_{qaud}^{sb} \left(\|\mathcal{R}_{sb}\|_{C^k}\right)\right)^{\frac{1}{2}} N_{sb}^{-\frac{\alpha_{sb}}{2}}\right]
\end{aligned}$$

From definition of training errors (5.8) and (5.15) and the above inequality, we obtain the desired estimate (5.9).

□

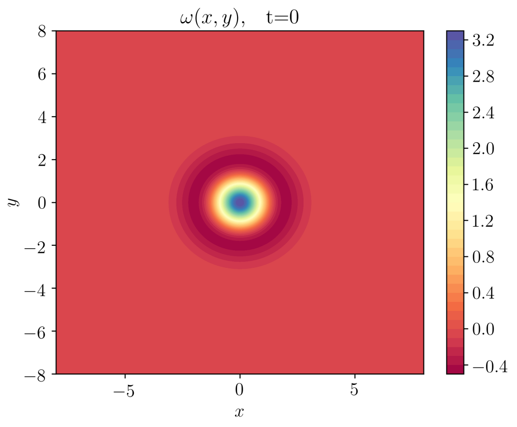
Remark 5.2. The bound (5.9) explicitly requires the existence of a classical solution $\mathbf{u}$ to the incompressible Euler equations, with a minimum regularity of $\nabla \mathbf{u} \in L^\infty(D \times (0,T))$. Such solutions do exist as long as we consider the incompressible Euler equations in *two space dimensions* and with sufficiently smooth initial data [30]. However, in three space dimensions, even with smooth initial data, the existence of smooth solutions is a major open question. It is possible that the derivative blows up and the constant C_∞ in (5.9) is unbounded, leading to a loss of control on the generalization error. In general, complicated solutions of the Euler equations are characterized by strong vorticity, resulting in large values of the spatial derivative. The bound (5.9) makes it clear that the generalization error with PINNs can be large for such problems. ■

5.4 Numerical Experiments

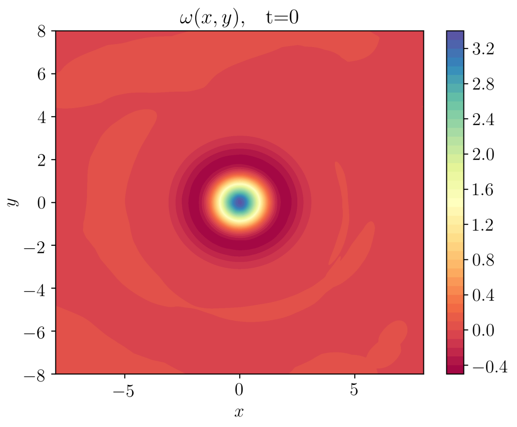
We present experiments for the incompressible Euler equations in two space dimensions, i.e $d = 2$ in (5.1). Moreover, we will use the *CELU* function given by,

$$CELU(x) = \max(0, x) + \min\big(0, \exp(x) - 1\big), \tag{5.16}$$

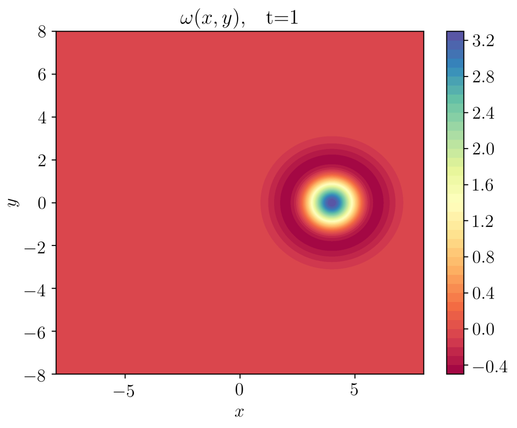
as the activation function σ in (2.8). The CELU function results in better approximation than the hyperbolic tangent, for the Euler equations.



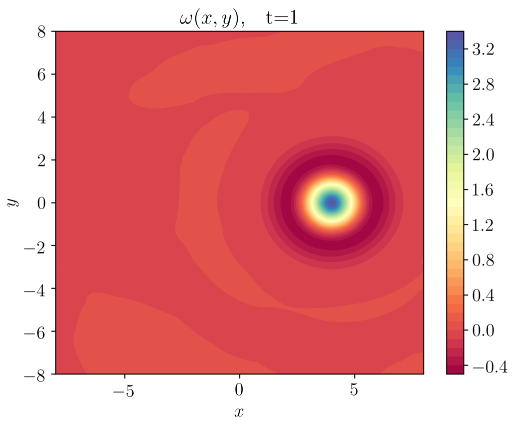
(a) Exact vorticity at $T = 0$



(b) Approximate (PINN) vorticity at $T = 0$



(c) Exact vorticity at $T = 1$



(d) Approximate (PINN) vorticity at $T = 1$

Figure 8: Exact and PINN solutions to the Taylor Vortex

5.4.1 Taylor Vortex

In the first numerical experiment, we consider the well-known Taylor Vortex, in a computational domain $D = [-8, 8]^2$ with periodic boundary conditions and with the initial conditions,

$$\begin{aligned} u_0(x, y) &= -ye^{\frac{1}{2}(1-x^2-y^2)} + a_x, \qquad (x, y) \in [-8, 8]^2, \\ v_0(x, y) &= xe^{\frac{1}{2}(1-x^2-y^2)} + a_y, \qquad (x, y) \in [-8, 8]^2, \end{aligned} \tag{5.17}$$

with $a_x = 8$ and $a_y = 0$.

In this case, one can obtain the following exact solution,

$$\begin{aligned} u(t, x, y) &= -(y - a_y t)e^{\frac{1}{2}\left[1-(x-a_x t)^2-(y-a_y t)^2\right]} + a_x, \\ v(t, x, y) &= (x - a_x t)e^{\frac{1}{2}\left[1-(x-a_x t)^2-(y-a_y t)^2\right]} + a_y. \end{aligned} \tag{5.18}$$

We will generate the training set with $N_{int} = 8192$, $N_{tb} = N_{sb} = 256$ points, chosen as low-discrepancy Sobol sequences on the underlying domains. An ensemble training procedure is performed, as described in the previous section, and resulted in the hyperparameter configuration presented in Table 5.

To visualize the solution, we follow standard practice and compute the vorticity $\omega = \text{curl}(\mathbf{u})$ and present the exact vorticity and the one obtained from the PINN, generated by algorithm 2.3 in figure 8. We remark that the vorticity can be readily computed from the PINN $\mathbf{u}^*$ by automatic differentiation. We see from the figure, that the PINN, approximates the flow field very well, both initially as well as at later times, with small numerical errors. This good quality of approximation is further reinforced by the generalization error (5.7), computed from (5.18) with 10^5 uniformly distributed random points, and presented in Table 5. We see that the generalization error for the best hyperparameter configuration is only 0.012%, indicating very high accuracy of the approximation for this test problem.

	N_{int}	N_{sb}	N_{tb}	$K-1$	d	L^1-reg.	L^2-reg.	λ	$\mathcal{E}_T$	$\mathcal{E}_G^r$
Taylor Vortex	8192	256	256	12	24	0.0	0.0	1	0.0003	0.012%
Double Shear Layer	65536	16384	16384	24	48	0.0	0.0	0.1	0.0025	3.8%

Table 5: Best performing *Neural Network* configurations for the Taylor Vortex and Double Shear Layer problem. Low-discrepancy Sobol points are used for every reported numerical example.

5.4.2 Double shear Layer

We consider the two-dimensional Euler equations (5.1) in the computational domain $D = [0, 2\pi]^2$ with periodic boundary conditions and consider initial data with the underlying vorticity, shown in figure 9 (Top Left). This vorticity, corresponds to a velocity field that has been evolved with a standard second-order finite difference projection method, with the well-known double shear layer initial data [4], evolved till $T = 1$. We are interested in determining if we can train a PINN to match the solution for later times.

To this end, we acknowledge that the underlying solution is rather complicated (see figure 9 Top row) for the corresponding reference vorticity, and consists of fast moving sharp vortices. Moreover, the vorticity is high, implying from the bound (5.9), that the generalization errors with PINNs can be high in this case. Hence, we consider training sets with larger number of points than the previous experiment, by setting $N_{int} = 65536$ and $N_{tb} = N_{sb} = 16384$. The ensemble training procedure resulted in hyperparameters presented in Table 5.

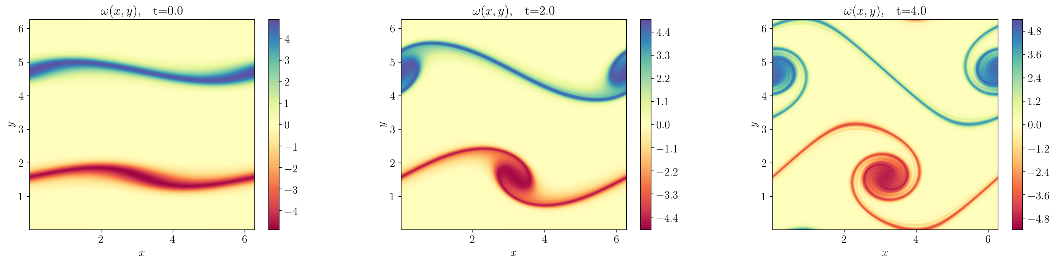
We present the approximate vorticity computed with the PINN, together with the exact vorticity, in figure 9, at three different times. From the figure, we see that the vorticity is approximated by the PINN quite well. However, the sharp vortices are smeared out and this is particularly apparent at later times. This is not surprising as the underlying solution is much more complicated in this case. Moreover, we have trained the PINN to approximate the velocity field, rather than the vorticity, and the generalization error (5.7) is still quite low at 3.8% (see Table 5).

Code

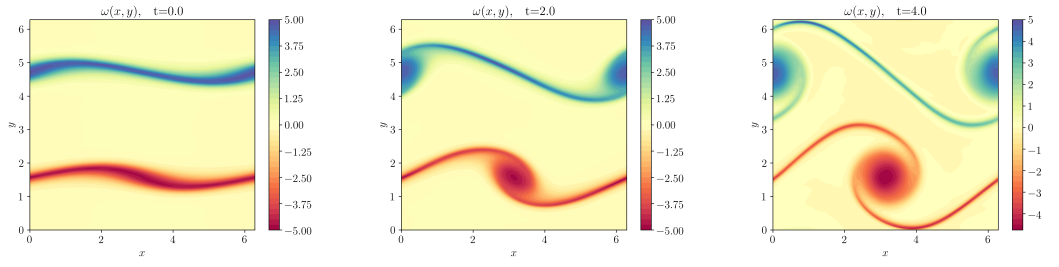
The building and the training of PINNs, together with the ensemble training for the selection of the model hyperparameters, are performed with a collection of Python scripts, realized with the support of PyTorch `https://pytorch.org/`. The scripts can be downloaded from `https://github.com/mroberto166/Pinns`.

6 Discussion

Physics informed neural networks (PINNs), originally proposed in [23], have recently been extensively used to numerically approximate solutions of PDEs in different contexts [38–40] and references therein. The main aim of this paper was to explain this efficient approximation by PINNs rigorously. We do so by proving bounds on the underlying generalization error i.e, error of the neural network on unseen data. To



(a) Reference, $T = 0$ (b) Reference, $T = 2$ (c) Reference, $T = 4$



(d) PINN, $T = 0$ (e) PINN, $T = 2$ (f) PINN, $T = 4$

Figure 9: Reference (Top Row) and PINN generated (Bottom Row) vorticities for the double shear layer problem at different times

this end, we introduce an abstract framework, where an abstract nonlinear PDE (2.1) is approximated by a PINN, generated with the algorithm 2.3. The key point of this algorithm was to minimize the PDE residual (2.12), at training points chosen as the quadrature points, corresponding to an underlying quadrature rule (2.6). The resulting generalization error is bounded in the abstract error estimate (2.21), in terms of the training error, the number of training samples and constants that stem from a stability estimates (2.3) for the underlying PDE. In particular for *well-trained networks* (2.23), the approximation by PINNs converges to the solution of the underlying PDE as the number of training samples is increased. Thus, we leverage the stability of nonlinear PDEs, together with accuracy of quadrature rules, in order to bound the generalization error for PINNs, providing a rigorous explanation for their robust performance.

We illustrate our approach with three representative examples for PDEs, i.e, semilinear parabolic PDEs, viscous scalar conservation laws and the incompressible Euler equations of fluid dynamics. For each of these examples, the abstract framework was worked out and resulted in the bounds (3.12), (4.8), (5.9) on the PINN generalization errors. All the bounds were of the form (2.21), in terms of the training and quadrature errors and with constants relying on stability (and regularity) estimates for *classical* solutions of the underlying PDE.

We also presented numerical experiments to validate the proposed theory. The numerical results were consistent with the derived estimates on the generalization error. For the heat equation, the results showed that PINNs are able to approximate the solutions accurately, even for very high (100) dimensional problems. For the viscous scalar conservation laws, we observed very accurate approximations with PINNs, for all values of the viscosity, as long as the underlying solution was at least Lipschitz continuous. However, as expected from the estimate (4.8), the accuracy deteriorated for the inviscid problem, when shocks were formed. Results with the two-dimensional incompressible Euler equations were also consistent with the derived error estimate (5.9).

With the exception of the very recent paper [43], this article is one of the first rigorous investigations on approximations of PDEs by PINNs. Given that [43] also considers this theme, it is instructive to compare the two articles and highlight differences. In [43], the authors focus on consistent approximation by PINNs by estimating the PINN loss in terms of the number of randomly chosen training samples and training error, corresponding to a *loss function*, regularized with Hölder norms of PDE residual, (Theorem 2.1 of [43]). Under assumptions of vanishing training error and uniform bounds on the residual in Hölder spaces, the authors prove convergence of the resulting PINN to the classical solution of the underlying PDE as the number of training samples increase. They illustrate their abstract framework on linear elliptic and parabolic PDEs.

Although similar in spirit, there are major differences in our approach compared to that of [43]. First, we do not need any additional regularization terms and directly estimate the generalization error from the stability estimate (2.3) and loss function (2.16). Convergence is proved to the underlying PDE solution as number of training samples increases (see lemma 2.7), under conditions on the training error (similar to assumption 3.3 (2) of [43]) and bounds on weights (similar to assumption 3.3 (3) of [43]). Second, given the abstract formalism of section 2, in particular the error estimate (2.21), we can cover very general PDEs, including non-linear PDEs. In fact, any PDE with a stability estimate of the form (2.3) is covered by our approach. We believe that this provides a unified explanation, in terms of stability estimates and quadrature rules, for the robust performance of PINNs for approximating a large number of linear and non-linear PDEs.

Finally, our current work has certain advantages as well as limitations and forms the foundation for the following extensions,

- We provided a very abstract framework for deriving error estimates for PINNs here and illustrated this approach for some representative linear and non-linear PDEs. Given the generality and universality of the proposed formalism, it is possible to extend the estimates for approximating a very wide class of PDEs that satisfy stability estimates of the form (2.3) and have classical solutions. In addition to the examples here, such PDEs include all well-known linear PDEs such as linear elliptic PDEs, wave equations, Maxwell's equations, linear elasticity, Stokes equations, Helmholtz equation etc as well as nonlinear PDEs such as Schrödinger equations, semilinear elliptic equations, dispersive equations such as KdV and many others. Extension of PINNs to these equations can be performed readily.

- A major limitation of the estimate (2.21) on the generalization error lies in the fact that the rhs in this estimate involves the training error (2.20). We are not able to estimate this error rigorously. However, this is standard in machine learning [29], where the training error is computed *a posteriori* (see the numerical results for training errors in respective experiments). The training error stems from the solution of a non-convex optimization problem in very high dimensions and there are no robust estimates on this problem. Once such estimates become available, they can be readily incorporated in the rhs of (2.21). For the time being, the estimate (2.21) should be interpreted as *as long as the PINN is trained well, it generalizes well.*

- Although never explicitly assumed, stability estimates of the abstract form (2.3), at least for nonlinear PDEs, amounts to regularity assumptions on the underlying solutions. For instance, the subspace Z in (2.3) is often a space of functions with sufficiently high Sobolev (or Hölder) regularity. The issue of regularity of the underlying PDE solutions is brought to the fore in the numerical example (see figure 6 and table 3) for the viscous Burgers' equation. Here, we clearly need the underling PDE solution to be Lipschitz continuous, for the PINN to appproximate it. Can PINNs approximate rough solutions of PDEs? The techniques introduced here may not suffice for this purpose and non-trivial extensions are needed.

- Finally, we have focused on the forward problems for PDEs in this paper. One of the most attractive features of PINNs is their ability to solve Inverse problems [31, 39, 40], with the same computational costs and efficiency as the forward problem. We prove rigorous generalization error estimates for PINNs, approximating inverse problems for PDEs in the companion paper [33].

Acknowledgements.

The research of SM and RM was partially supported by European Research Council Consolidator grant ERCCoG 770880: COMANFLO.

References

[1] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang. Stronger generalization bounds for deep nets via a compression approach. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 254–263, 2018.

[2] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inform. Theory.*, 39(3):930–945, 1993.

[3] C. Beck, S. Becker, P. Grohs, N. Jaafari, and A. Jentzen. Solving stochastic differential equations and kolmogorov equations by means of deep learning. Preprint, available as arXiv:1806.00421v1.

[4] J. Bell, P. Collela, and H. M. Glaz. A second-order projection method for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 85:257–283, 1989.

[5] H. J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004.

[6] R. E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta Numerica*, 7:1–49, 1998.

[7] J. Calder. Consistency of lipschitz learning with infinite unlabeled data and finite labeled data,. *SIAM Journal on Mathematics of Data Science*, 1:780–812, 2019.

[8] F. Cucker and S. Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39(1):1–49, 2002.

[9] G. Cybenko. Approximations by superpositions of sigmoidal functions. *Approximation theory and its applications.*, 9(3):17–28, 1989.

[10] R. Dautray and J. L. Lions. *Mathematical analysis and numerical methods for science and technology vol 5 Evolution equations I.* Springer Verlag, 1992.

[11] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[12] R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Zidek, A. Nelson, A. Bridgland, H. Penedones, et al. De novo structure prediction with deep-learning based scoring. *Annual Review of Biochemistry*, 77(6):363–382, 2018.

[13] L. D. F. Regazzoni and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational physics*, 397, 2019.

[14] R. Fletcher. *Practical methods of optimization.* John Wiley and sons, 1987.

[15] A. Friedman. *Partial differential equations of the parabolic type.* prentice hall, 1964.

[16] E. Godlewski and P. A. Raviart. *Hyperbolic systems of conservation laws.* Ellipsis, 1991.

[17] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning.* MIT press, 2016.

[18] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[19] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks.*, 2(5):359–366, 1989.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[21] F. Laakmann and P. Petersen. Efficient approximation of solutions of parametric linear transport equations by reludnns. Preprint, 2019.

[22] I. E. Lagaris, A. Likas, and P. G. D. Neural-network methods for bound- ary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11:1041–1049, 2000.

[23] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[24] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[25] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. Preprint, available from arXiv:1907.04502, 2019.

[26] K. . O. Lye, S. Mishra, P. Chandrasekhar, and D. Ray. Iterative surrogate model optimization (ismo): An active learning algorithm for pde constrained optimization with deep neural networks. Preprint,, 2020.

[27] K. O. Lye, S. Mishra, and R. Molinaro. A multi-level procedure for enhancing accuracy of machine learning algorithms. *European Journal of Applied Mathematics*, 2020.

[28] K. O. Lye, S. Mishra, and D. Ray. Deep learning observables in computational fluid dynamics. *Journal of Computational Physics*, page 109339, 2020.

[29] A. R. M. Mohri and A. Talwalkar. *Foundations of machine learning.* MIT press, 2018.

[30] A. J. Majda and A. L. Bertozzi. *Vorticity and Incompressible Flow.* Cambridge Texts in Applied Mathematics. Cambridge University Press, 2001.

[31] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[32] S. Mishra. A machine learning framework for data driven acceleration of computations of differential equations. *Mathematics in Engineering*, 1:118, 2019.

[33] S. Mishra and R. Molinaro. Estimates on the generalization error of physics informed neural networks (pinns) for approximating pdes ii: A class of inverse problems. Preprint, 2020.

[34] S. Mishra and T. K. Rusch. Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences. Preprint, available as arXiv:2005.12564, 2020.

[35] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro. The role of over-parametrization in generalization of neural networks. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[36] A. B. Owen. Multidimensional variation for quasi-monte carlo. In *Contemporary Multivariate Analysis And Design Of Experiments: In Celebration of Professor Kai-Tai Fang's 65th Birthday*, pages 49–74. World Scientific, 2005.

[37] G. Pang, L. Lu, and G. E. Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM journal of Scientific computing*, 41:A2603–A2626, 2019.

[38] M. Raissi and G. E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[39] M. Raissi, P. Perdikaris, and 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*, 378:686–707, 2019.

[40] M. Raissi, A. Yazdani, and G. E. Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *arXiv preprint arXiv:1808.04327*, 2018.

[41] D. Ray and J. S. Hesthaven. An artificial neural network as a troubled-cell indicator. *Journal of Computational Physics*, 367:166–191, 2018.

[42] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge University Press, 2014.

[43] Y. Shin, J. Darbon, and G. E. Karniadakis. On the convergence and generalization of physics informed neural networks. Preprint, available from arXiv:2004.01806v1, 2020.

[44] J. Stoer and R. Bulirsch. *Introduction to numerical analysis.* Springer Verlag, 2002.

[45] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.