DPM: A Novel Training Method for Physics-Informed Neural Networks in Extrapolation

Jungeun Kim[1], **Kookjin Lee**[2], **Dongeun Lee**[3], **Sheo Yon Jhin**[1], **Noseong Park**[4]
[1]Department of AI, Yonsei University
[2]Extreme Scale Data Science & Analytics Department, Sandia National Laboratory
[3]Department of Computer Science and Information Systems, Texas A&M University at Commerce
[4]Department of AI & CS, Yonsei University
jekim5418@yonsei.ac.kr, koolee@sandia.gov, dongeun.lee@tamuc.edu, {sheoyonj, noseong}@yonsei.ac.kr

Abstract

We present a method for learning dynamics of complex physical processes described by time-dependent nonlinear partial differential equations (PDEs). Our particular interest lies in extrapolating solutions in time beyond the range of temporal domain used in training. Our choice for a baseline method is physics-informed neural network (PINN) because the method parameterizes not only the solutions, but also the equations that describe the dynamics of physical processes. We demonstrate that PINN performs poorly on extrapolation tasks in many benchmark problems. To address this, we propose a novel method for better training PINN and demonstrate that our newly enhanced PINNs can accurately extrapolate solutions in time. Our method shows up to 72% smaller errors than existing methods in terms of the standard L2-norm metric.

1 Introduction

Understanding dynamics of complex real-world physical processes is essential in many applications (e.g., fluid dynamics (Anderson, Tannehill, and Pletcher 2016; Hirsch 2007)). Such dynamics are often modeled as time-dependent partial deferential equations (PDEs), where we seek a solution function $u(x,t)$ satisfying a governing equation,

$$f(x,t) \stackrel{\text{def}}{=} u_t + \mathcal{N}(u) = 0,\ x \in \Omega,\ t \in [0,T], \qquad (1)$$

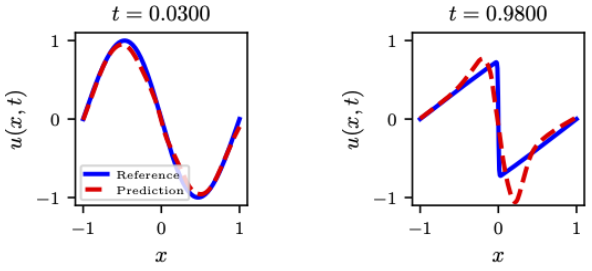
where $u_t \stackrel{\text{def}}{=} \frac{\partial u}{\partial t}$ denotes the partial derivative of u w.r.t. t, $\mathcal{N}$ denotes a nonlinear differential operator, $\Omega \subset \mathbb{R}^d$ ($d = 1, 2, 3$) denotes a spatial domain, and T denotes the final time. Moreover, there are two more types of conditions imposed on the solution function $u(x,t)$: i) an initial condition $u(x,0) = u^0(x), \forall x \in \Omega$ and ii) a set of boundary conditions specifying the behaviors of the solution function on the boundaries of Ω. Solving such problem becomes particularly challenging when the nonlinear differential operator is highly nonlinear.

Traditionally, classical numerical methods (e.g., (Iserles 2009; LeVeque et al. 2002; Stoer and Bulirsch 2013)) have been dominant choices for solving such nonlinear time-dependent PDEs, as they have demonstrated their effectiveness in solving complex nonlinear PDEs and provide sound theoretical analyses. However, they are often based on techniques that require complex, problem-specific knowledge such as sophisticated spatio-temporal discretization schemes.

Recently, with the advancements in deep learning, many data-centric approaches, which heavily rely on the universal approximation theorem (Hornik et al. 1989), have been proposed. Most approaches formulate the problem as a rather simple (semi-)supervised learning problem for constructing surrogate models for solution functions (Geist et al. 2020; Khoo, Lu, and Ying 2017; Ling, Kurzawski, and Templeton 2016; Ling and Templeton 2015; Tripathy and Bilionis 2018; Vlachas et al. 2018; Holl, Thuerey, and Koltun 2020). Although the formulation itself is simple, this approach requires costly evaluations or existence of solutions and also suffers from lack of ways to enforce a priori information of the problem such as physical laws described by the governing equation. There are also more "physics-aware" approaches such as methods based on learning latent-dynamics of physical processes (Erichson, Muehlebach, and Mahoney 2019; Fulton et al. 2019; Lee and Carlberg 2019, 2020; Wiewel, Becher, and Thuerey 2019). These approaches, however, still require computations of solutions to collect training dataset.

Among those data-centric approaches, a method called physics-informed neural network (PINN) (Raissi, Perdikaris, and Karniadakis 2019) has brought attention to the community because of its simple, but effective way of approximating time-dependent nonlinear PDEs with neural networks, while preserving important physical properties described by the governing equations. PINN achieves these by parameterizing the solution and the governing equation simultaneously with a set of shared network parameters, which we will elaborate in the next section. After the great success of the seminal paper (Raissi, Perdikaris, and Karniadakis 2019), many sequels have applied PINN to solve various PDE applications, e.g. (Cosmin Anitescu 2019; Yang, Meng, and Karniadakis 2020; Zhang et al. 2019; Doan, Polifke, and Magri 2019).

Nearly all these studies, however, demonstrated the performances of their methods evaluated at a set of testing points sampled within a pre-specified range, i.e., $\{(x^i_{\text{test}}, t^i_{\text{test}})\} \subset \Omega \times [0, T_{\text{train}}] \setminus \{(x^i_{\text{train}}, t^i_{\text{train}})\}$, which we denote by *interpolation*. In this paper, however, we are more interested in assessing the capability of PINN as a tool for learning the dynamics of physical processes. In particular, we would like to assess the performance of PINN on a testing set sampled beyond the final training time T_{train} of the pre-specified range, i.e., $\{(x^i_{\text{test}}, t^i_{\text{test}})\} \subset \Omega \times (T_{\text{train}}, T]$, where $T > T_{\text{train}}$, and we denote this task by *extrapolation*. In principle, PINN is ex-



(a) Interpolation (b) Extrapolation

Figure 1: 1D viscous Burgers' equation examples. We train the PINN model (Raissi, Perdikaris, and Karniadakis 2019) with $T_{\text{train}} = 0.5$ and report two solution snapshots of the reference solution (solid blue line) and the approximated solution (dashed red line) obtained by PINN at $t = 0.03$ (i.e., interpolation) and $t = 0.98$ (i.e., extrapolation).

pected to learn the dynamics Eq. (1) and, consequently, to approximate $u(x,t)$ in $(T_{\text{train}}, T]$ accurately if trained properly. However, in our preliminary study with a one-dimensional viscous Burgers' equation shown in Fig. 1, we observe that the accuracy of the approximate solution produced by PINN in the extrapolation setting is significantly degraded compared to that produced in the interpolation setting.

Motivated by this observation, we analyze PINN in detail (Section 2), propose our method to improve the approximation accuracy in extrapolation (Section 3), and demonstrate the effectiveness of the proposed method with various benchmark problems (Section 4). In all benchmark problems, our proposed methods, denoted by PINN-D1 and D2, show the best accuracies with various evaluation metrics. In comparison with state-of-the-art methods, errors from our proposed methods are up to 72% smaller.

2 Related Work and Preliminaries

We now formally introduce PINN. Essentially, PINN parameterizes both the solution u and the governing equation f. Let us denote a neural network approximation of the solution $u(x,t)$ by $\tilde{u}(x,t;\Theta)$, where Θ denotes a set of network parameters. The governing equation f is then approximated by a neural network $\tilde{f}(x,t,\tilde{u};\Theta) \stackrel{\text{def}}{=} \tilde{u}_t + \mathcal{N}(\tilde{u}(x,t;\Theta))$, where partial derivatives are obtained via automatic differentiation (or a back-propagation algorithm (Rumelhart, Hinton, and Williams 1986) to be more specific). That is, the neural network $\tilde{f}(x,t,\tilde{u};\Theta)$ shares the same network weights with $\tilde{u}(x,t;\Theta)$, but enforces physical laws by applying an extra problem-specific nonlinear activation defined by the PDE in Eq. (1) (i.e., $\tilde{u}_t + \mathcal{N}(\tilde{u})$), which leads to the name "physics-informed" neural network.[1]

This construction suggests that these shared network weights can be learned via forming a loss function consisting of two terms, each of which is associated with approximation errors in $\tilde{u}$ and $\tilde{f}$, respectively. In the original formulation, a loss function consisting of two error terms is considered:

$$L \stackrel{\text{def}}{=} \alpha L_u + \beta L_f, \tag{2}$$

where $\alpha, \beta \in \mathbb{R}$ are coefficients and L_u, L_f are defined below.

$$L_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(x_u^i, t_u^i) - \tilde{u}(x_u^i, t_u^i; \Theta)|^2, \tag{3}$$

$$L_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |\tilde{f}(x_f^i, t_f^i, \tilde{u}; \Theta)|^2. \tag{4}$$

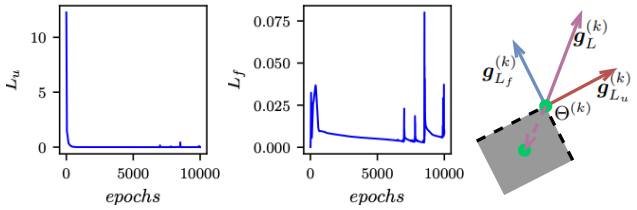
The first loss term, L_u, enforces initial and boundary conditions using a set of training data $\left\{\left((x_u^i, t_u^i), u(x_u^i, t_u^i)\right)\right\}_{i=1}^{N_u}$, where the first element of the tuple is the input to the neural network $\tilde{u}$ and the second element is the ground truth that the output of $\tilde{u}$ attempts to match. These data can be easily collected from specified initial and boundary conditions, which are known *a priori* (e.g., $u(x,0) = u^0(x) = -\sin(\pi x)$ in a PDE we use for our experiments). The second loss term, L_f, minimizes the discrepancy between the governing equation f and the neural network approximation $\tilde{f}$ evaluated at *collocation points*, which forms another training dataset $\left\{\left((x_f^i, t_f^i), f(x_f^i, t_f^i)\right)\right\}_{i=1}^{N_f}$, where the ground truth $\{f(x_f^i, t_f^i)\}_{i=1}^{N_f}$ consists of all zeros.

The advantages of this loss construction are that i) no costly evaluations of the solutions $u(x,t)$ at collocation points are required to collect training data, ii) initial and boundary conditions are enforced by the first loss term L_u where its training dataset can be easily generated, and iii) a physical law described by the governing equation f in Eq. (1) can be enforced by minimizing the second loss term L_f. In (Raissi, Perdikaris, and Karniadakis 2019), both the loss terms are considered equally important (i.e., $\alpha = \beta = 1$), and the combined loss term L is minimized.

Motivations. If PINN can correctly learn a governing equation, its extrapolation should be as good as interpolation. Successful extrapolation will enable the adoption of PINN to many PDE applications. With the loss formulation in Eq. (2), however, we empirically found that it is challenging to train PINN for extrapolation as shown in Fig. 1.

Hence, we first investigate training loss curves of L_u and L_f separately: Fig. 2 depicts the loss curves L_u and L_f of PINN trained for a 1D inviscid Burgers' equation. The figure shows that L_u converges very fast, whereas L_f starts to fluctuate after a certain epoch and does not decrease below a certain value. From the observation, we can conclude that the initial and the boundary conditions are successfully enforced, whereas the dynamics of the physical process may not be accurately enforced, which, consequently, could lead to significantly less accurate approximations in extrapolation, e.g., Fig. 1(b). Motivated by this observation, we propose a novel training method for PINN in the following section. In the experiments section, we demonstrate performances of the proposed training method.

[1] We also note that there are other studies (e.g., (Cranmer et al. 2020; Greydanus, Dzamba, and Yosinski 2019)) using the idea of parameterizing the governing equations, where derivatives are also computed using automatic differentiation.



(a) L_u curve (b) L_f curve (c) Updating Θ

Figure 2: Example training curves of L_u and L_f of PINN for a 1D inviscid Burgers' equation in (a) and (b) respectively, and an example of updating Θ in (c)

3 Dynamic Pulling Method (DPM)

The issue with training PINN, which we have identified in our preliminary experiments, is that L_f is fluctuating and is not decreasing. To resolve this issue, we propose a novel training method to impose a *soft* constraint of $L_f \leq \epsilon$, where ϵ is a hyperparameter and can be set to an arbitrary small value to ensure an accurate approximation of the governing equation, i.e., enforcing $\tilde{f}(\cdot)$ to be close to zero. The proposed training concept is dynamically manipulating the gradients.

We dynamically manipulate the gradients of the loss terms on top of a gradient-based optimizer including but not limited to the gradient descent method, i.e., $\Theta^{(k+1)} = \Theta^{(k)} - \gamma \boldsymbol{g}^{(k)}$, where γ is a learning rate, and $\boldsymbol{g}^{(k)}$ is a gradient at k-th epoch. We set the gradient $\boldsymbol{g}^{(k)}$ to one of the following vectors depending on conditions:

$$\boldsymbol{g}^{(k)} = \begin{cases} \boldsymbol{g}_{L_u}^{(k)} & \text{, if } L_f \leq \epsilon \\ \boldsymbol{g}_L^{(k)} & \text{, if } L_f > \epsilon \wedge \boldsymbol{g}_{L_u}^{(k)} \cdot \boldsymbol{g}_{L_f}^{(k)} \geq 0, \\ \boldsymbol{v} + \boldsymbol{g}_L^{(k)} & \text{, otherwise} \end{cases} \quad (5)$$

where $\boldsymbol{v} \in \mathbb{R}^{\dim(\Theta)}$ is a manipulation vector, which we will show how to calculate shortly; $\boldsymbol{g}_{L_u}^{(k)}$, $\boldsymbol{g}_{L_f}^{(k)}$, and $\boldsymbol{g}_L^{(k)}$ denote the gradients of L_u, L_f, and L, respectively.

Here, we care only about $\boldsymbol{g}_{L_u}^{(k)}$, when L_f is small enough, i.e., $L_f \leq \epsilon$, because L_f already satisfies the constraint. There are two possible cases when $L_f > \epsilon$: i) $\boldsymbol{g}_{L_u}^{(k)} \cdot \boldsymbol{g}_{L_f}^{(k)} \geq 0$ and ii) $\boldsymbol{g}_{L_u}^{(k)} \cdot \boldsymbol{g}_{L_f}^{(k)} < 0$. In the former case where the two gradient terms $\boldsymbol{g}_{L_u}^{(k)}$ and $\boldsymbol{g}_{L_f}^{(k)}$ have the same direction (i.e., the angle between them is less than $90°$ and hence their dot-product is positive), performing a gradient descent update with $\boldsymbol{g}_L^{(k)}$ guarantees a decrease in L_f. In Fig. 2 (c), for instance, both L_f and L_u decrease if $\Theta^{(k)}$ is updated into the gray area.

When $L_f > \epsilon$ and $\boldsymbol{g}_{L_u}^{(k)} \cdot \boldsymbol{g}_{L_f}^{(k)} < 0$, however, $\boldsymbol{v}$ carefully manipulates the gradient in such a way that L_f is guaranteed to decrease after a gradient descent update.

We now seek such a solution $\boldsymbol{v}$ that will result in $\left(\boldsymbol{v} + \boldsymbol{g}_L^{(k)}\right) \cdot \boldsymbol{g}_{L_f}^{(k)} > 0$ given $\boldsymbol{g}_L^{(k)}$ and $\boldsymbol{g}_{L_f}^{(k)}$. Because the dot-product is distributive, it satisfies the following condition

$$\left(\boldsymbol{v} + \boldsymbol{g}_L^{(k)}\right) \cdot \boldsymbol{g}_{L_f}^{(k)} = \boldsymbol{v} \cdot \boldsymbol{g}_{L_f}^{(k)} + \boldsymbol{g}_L^{(k)} \cdot \boldsymbol{g}_{L_f}^{(k)} > 0, \quad (6)$$

which can be re-formulated as follows:

$$\boldsymbol{v} \cdot \boldsymbol{g}_{L_f}^{(k)} + \boldsymbol{g}_L^{(k)} \cdot \boldsymbol{g}_{L_f}^{(k)} = \delta, \quad (7)$$

where $\delta > 0$ is to control how much we pull $\Theta^{(k)}$ toward the region where L_f decreases, e.g., the gray region of Fig. 2 (c).

We note that Eq. (7) has many possible solutions. Among them, one solution, denoted $\boldsymbol{v}^* = \frac{-\boldsymbol{g}_L^{(k)} \cdot \boldsymbol{g}_{L_f}^{(k)} + \delta}{\|\boldsymbol{g}_{L_f}^{(k)}\|_2^2} \boldsymbol{g}_{L_f}^{(k)}$, can be computed by using the *pseudoinverse* of $\boldsymbol{g}_{L_f}^{(k)}$, which is widely used to find such solutions, e.g., the analytic solution of linear least-squared problems arising in linear regressions.

A good characteristic of the pseudoinverse is that it minimizes $\|\boldsymbol{v}\|_2^2$ (Ben-Israel and Greville 2006). By minimizing $\|\boldsymbol{v}\|_2^2$, we can disturb the original updating process as little as possible. Therefore, we use the pseudoinverse-based solution in our method.

Despite its advantage, the gradient manipulation vector $\boldsymbol{v}^*$ sometimes requires many iterations until $L_f \leq \epsilon$. To expedite the pulling procedure, we also dynamically control the additive pulling term δ as follows:

$$\Delta^{(k)} = L_f(\Theta^{(k)}) - \epsilon, \quad (8)$$

$$\delta^{(k+1)} = \begin{cases} w\delta^{(k)}, & \text{if } \Delta^{(k)} > 0, \\ \frac{\delta^{(k)}}{w}, & \text{if } \Delta^{(k)} \leq 0, \end{cases} \quad (9)$$

where $w > 1$ is an inflation factor for increasing δ.

4 Experiments

We describe our experimental environments and results with four benchmark time-dependent nonlinear PDEs and several different neural network designs. Our software and hardware environments are as follows: UBUNTU 18.04 LTS, PYTHON 3.6.6, NUMPY 1.18.5, SCIPY 1.5, MATPLOTLIB 3.3.1, TENSORFLOW-GPU 1.14, CUDA 10.0, and NVIDIA Driver 417.22, i9 CPU, and NVIDIA RTX TITAN.

4.1 Experimental Environments

PDEs. We consider viscous and inviscid Burgers' equations, nonlinear Schrödinger equation (NLS), and Allen–Cahn (AC) equation. We refer readers to the full version (Kim et al. 2020) for detailed descriptions for these equations.

For training/validating/testing, we divide the entire time domain $[0, T]$ into three segments: $[0, T_{\text{train}}]$, $(T_{\text{train}}, T_{\text{val}}]$, and $(T_{\text{val}}, T_{\text{test}}]$, where $T = T_{\text{test}} > T_{\text{val}} > T_{\text{train}} > 0$. In other words, our task is to predict the solution functions of the PDEs in a future time frame, i.e., extrapolation. We use $T_{\text{train}} = \frac{T}{2}$, $T_{\text{val}} = \frac{4T}{5}$, and $T_{\text{test}} = T$, i.e., extrapolating for the last 20% of the time domain, which is a widely used setting in many time-series prediction studies (Kim 2003; Kang et al. 2016).

PDE	L2-norm (↓)				Explained variance score (↑)				Max error (↓)				Mean absolute error (↓)			
	PINN	PINN-R	PINN-D1	PINN-D2	PINN	PINN-R	PINN-D1	PINN-D2	PINN	PINN-R	PINN-D1	PINN-D2	PINN	PINN-R	PINN-D1	PINN-D2
Vis. Burgers	0.329	0.333	0.106	**0.092**	0.891	0.901	0.988	**0.991**	0.657	1.081	0.545	**0.333**	0.085	0.108	0.026	**0.021**
Inv. Burgers	0.131	0.095	**0.083**	0.090	0.214	0.468	0.485	**0.621**	3.088	2.589	**1.534**	2.036	0.431	0.299	**0.277**	0.315
Allen–Cahn	0.350	0.286	0.246	**0.182**	0.090	0.919	0.939	**0.967**	1.190	1.631	1.096	**0.836**	0.212	0.142	0.129	**0.094**
Schrödinger	0.239	0.212	0.314	**0.141**	-4.364	-3.902	-4.973	**-3.257**	4.656	4.222	4.945	**3.829**	0.954	0.894	**0.868**	0.896

Table 1: The extrapolation accuracy in terms of the relative errors in the L2-norm, the explained variance error, the max error, and the mean absolute error in various PDEs. Large (resp. small) values are preferred for ↑ (resp. ↓).

Baselines. Our task definition is not to simply approximate a solution function u with a regression model but to let a neural network learn physical dynamics without costly collections of training samples (see our broader impact statement to learn why it is important to train without costly collections of training samples). For this task, the state-of-the-art method is PINN. We compare our method with the following baselines: i) the original PINN which uses a series of fully-connected and hyperbolic tangent layer, denoted by PINN, and ii) PINN improved with the residual connection (He et al. 2016), denoted by PINN-R. We apply our DPM with (resp. without) controlling δ in Eq. (9) to train PINN-R, denoted by PINN-D2 (resp. PINN-D1).

Evaluation Metrics. For performance evaluation, we collect predicted solutions at testing data instances to construct a solution vector $\tilde{\boldsymbol{u}} = [\tilde{u}(x^1_{\text{test}}, t^1_{\text{test}}; \Theta), \tilde{u}(x^2_{\text{test}}, t^2_{\text{test}}; \Theta), \ldots]^\top$, where $\{(x^i_{\text{test}}, t^i_{\text{test}})\}$ is a set of testing samples. x^i_{test} is sampled at a uniform spatial mesh grid in Ω and t^i_{test} is on a uniform temporal grid in $(T_{\text{val}}, T_{\text{test}}]$. See the full version (Kim et al. 2020) for how to build testing sets. For the comparison, we also collect the reference solution vector, denoted $\boldsymbol{u}$, at the same testing data instances by solving the same PDEs using traditional numerical solvers. As evaluation metrics, we use the standard relative errors in L2-norm, i.e., $\|\tilde{\boldsymbol{u}} - \boldsymbol{u}\|_2 / \|\boldsymbol{u}\|_2$, the explained variance score, the max error, and the mean absolute error, each of which shows a different aspect of performance. Moreover, we report snapshots of the reference and approximate solutions at certain time indices.

Hyperparameters. For all methods, we test with the following hyperparameter configurations: the number of layers is $\{2, 3, 4, 5, 6, 7, 8\}$, the dimensionality of hidden vector is $\{20, 40, 50, 100, 150\}$. For PINN and PINN-R, we use $\alpha = \{1, 10, 100, 1000\}$, $\beta = \{1, 10, 100, 1000\}$ — we do not test the condition of $\alpha = \beta$, except for $\alpha = \beta = 1$. Our DPM uses $\alpha = \beta = 1$. The learning rate is {1e-3, 5e-3, 1e-4, 5e-5} with various standard optimizers such as Adam, SGD, etc. For the proposed DPM, we test with $\epsilon = \{0.001, 0.005, 0.01, 0.0125\}$, $\delta = \{0.01, 0.1, 1, 10\}$, and $w = \{1.001, 1.005, 1.01, 1.025\}$. We also use the early stopping technique using the validation error as a criterion. If there are no improvements in validation loss larger than 1e-5 for the past 50 epochs, we stop the training process. We choose the model that best performs on the validation set.

Train & Test Set Creation. To build testing sets, x^i_{test} is sampled at a uniform spatial mesh grid in Ω and t^i_{test} is on a uniform temporal grid in $(T_{\text{val}}, T_{\text{test}}]$. We use a temporal step size of 0.01, 0.0175, $\frac{0.01\pi}{2}$, and 0.005 for the viscous Burgers' equation, the inviscid Burgers' equation, the NLS equation, and the AC equation, respectively. We divide Ω into a grid of 256, 512, 256, and 256 points for the viscous Burgers' equation, the inviscid Burgers' equation, the NLS equation, and the AC equation, respectively.

For creating our training sets, we use $N_u = 100$ initial and boundary tuples for all the benchmark equations. For N_f, we use 10K for the viscous and the inviscid Burgers' equations, and 20K for the NLS equation and the AC equation.

4.2 Experimental Results

Table 1 summarizes the overall performance for all benchmark PDEs obtained by PINN, PINN-R, PINN-D1, and PINN-D2. PINN-R shows smaller L2-norm errors than PINN. The proposed PINN-D2 significantly outperforms PINN and PINN-R in all four benchmark problems for all metrics. For the viscous Burgers' equation and the AC equation, PINN-D2 demonstrates 72% and 48% (resp. 72% and 36%) improvements over PINN (resp. PINN-R) in terms of the relative L2-norm, respectively.

Viscous Burgers' equation. Fig. 3 shows the reference solution and predictions made by PINN and the PINN variants of the viscous Burgers' equation. In Figs. 3(b)–3(c), both PINN and PINN-R fail to correctly learn the governing equation and their prediction accuracy is significantly degraded as t increases. However, the proposed PINN-D2 shows much more accurate prediction even when t is close to the end of the time domain. These results explain that learning a governing equation correctly helps accurate extrapolation. Although PINN and PINN-R are able to learn the initial and boundary conditions accurately, their extrapolation performances are poor because they fail to learn the governing equation accurately. Figs. 3(e)–3(j) report solution snapshots at $t = \{0.83, 0.98\}$ and we observe that the proposed PINN-D2 outperforms the other two PINN methods. Only PINN-D2 accurately enforces the prediction around $x = 0$ in Fig. 3(j). PINN-D1 is comparable to PINN-D2 in this equation according to Table 1.

Inviscid Burgers' equation. In this benchmark problem, we consider the inviscid Burgers' equation posed on a very long time domain $[0, 35]$, which is much larger than those of other benchmark problems and could make the extrapolation task even more challeging. Fig. 4 reports the results obtained by the PINN variants along with the reference solution. All

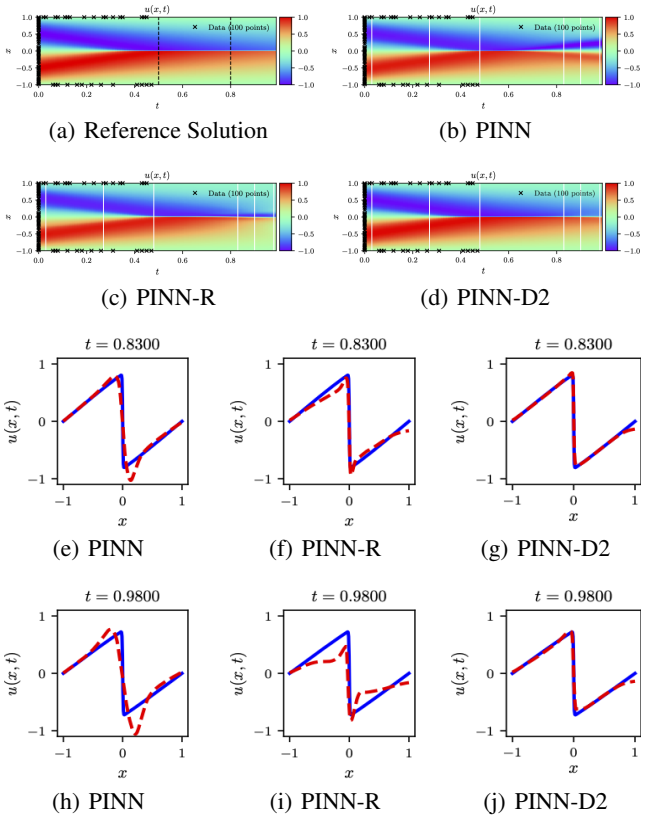


Figure 3: Top two rows: the complete reference solution and predictions of the benchmark viscous Burgers' equation. The points marked with $\times$ mean initial or boundary points. Bottom: the solution snapshots at $t = \{0.83, 0.98\}$ obtained via the extrapolation. In Fig. 3(a), the black vertical lines correspond to T_{train} and T_{val}, respectively, and in Figs. 3(b)–3(d), the white vertical lines correspond to time indices, where we extract solution snapshots. We refer readers to the full version (Kim et al. 2020) for more snapshots. The meanings of the vertical lines remain the same in the following figures.

the three methods, PINN-R, PINN-D1, and PINN-D2, are comparable in this benchmark problem. However, we can still observe that PINN-D2 produces slightly more accurate predictions than other methods at $x = 0$, the boundary condition. The first condition of Eq. (5) accounts for this result: when L_f is sufficiently small, the update performed by DPM further decreases L_u to improve the predictions in the initial and boundary conditions.

Allen–Cahn equation (AC). Fig. 5 reports the reference solutions of the AC equation and the predictions made by all the considered PINN variants. The solution snapshots shown in Figs. 5(e)–5(j) demonstrate that the proposed PNN-D2 produces the most accurate approximations to the reference solutions. In particular, the approximate solutions obtained by using PINN-D2 matches very closely with the reference solutions with the exception on the valley (around $x = 0$),

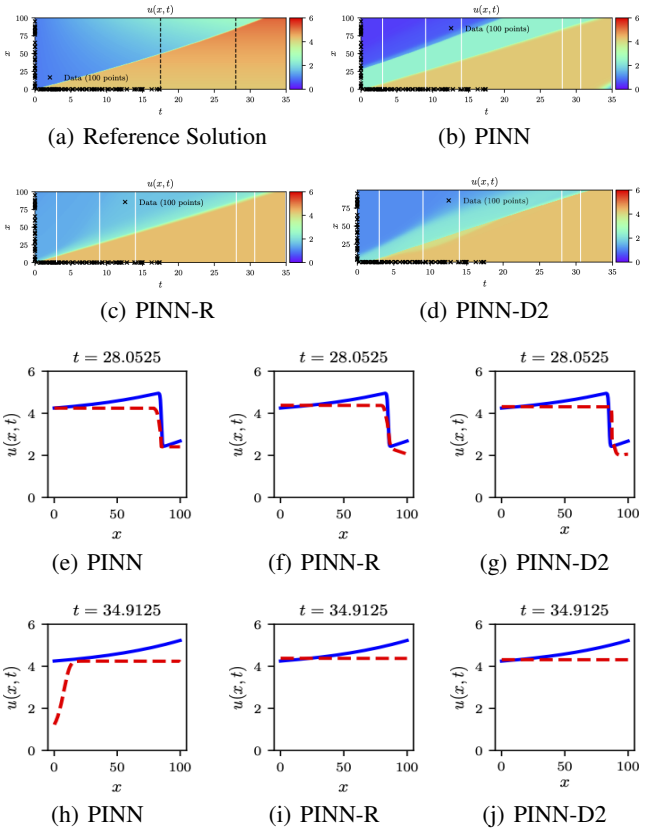


Figure 4: Top two rows: the complete reference solution and predictions of the benchmark inviscid Burgers' equation. The points marked with $\times$ mean initial or boundary points. Bottom: the solution snapshots at $t = \{28.0875, 34.9125\}$ obtained via the extrapolation.

where all three methods struggle to make accurate predictions. Moreover, the approximate solutions of PINN-D2 are almost symmetric w.r.t. $x = 0$, whereas the approximate solutions of the other two methods are significantly non-symmetric and the accuracy becomes even more degraded as t increases.

Nonlinear Schrödinger equation (NLS). Fig. 6 reports the reference solution of the NLS equation and the predictions made by all the considered PINN variants. Because the solution of the NLS equation is a complex-valued, the magnitudes of the reference solution $|u(x,t)|$ and the predictions $|\tilde{u}(x,t)|$ are depicted. The solution snapshots produced by PINN and PINN-R exhibit errors around $x = -1$ and $x = 1$ whereas PINN-D2 is accurate around the region. In particular, the predictions made by PINN and PINN-R exhibit the shapes that are very similar to previous time steps' solution snapshots, which indicates that the dynamics of the system is not learned accurately. In contrast, PINN-D2 seems to enforce the dynamics much better and produce more accurate predictions.

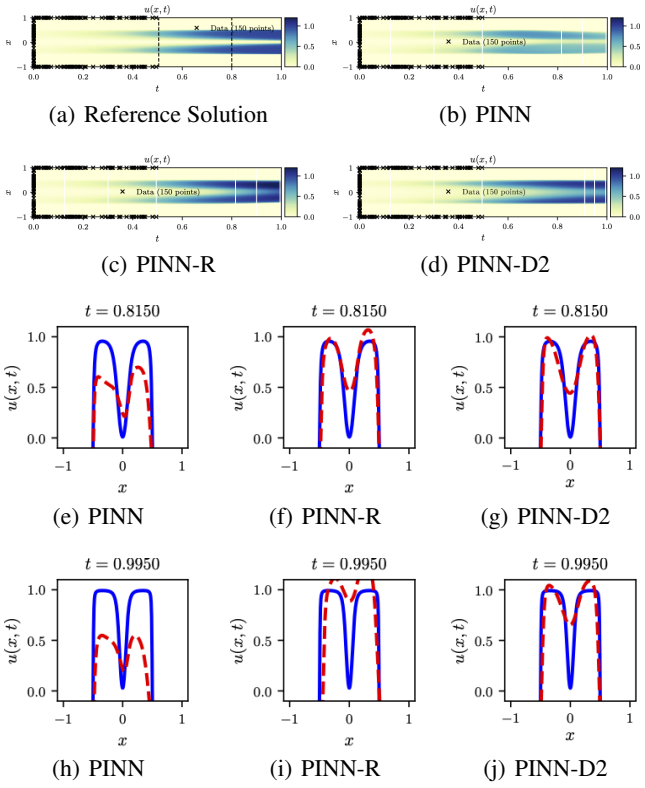


Figure 5: Top two rows: the complete reference solution and predictions of the Allen–Cahn equation. The points marked with $\times$ mean initial or boundary points. Bottom: the extrapolation solution snapshots at $t = \{0.815, 0.995\}$.

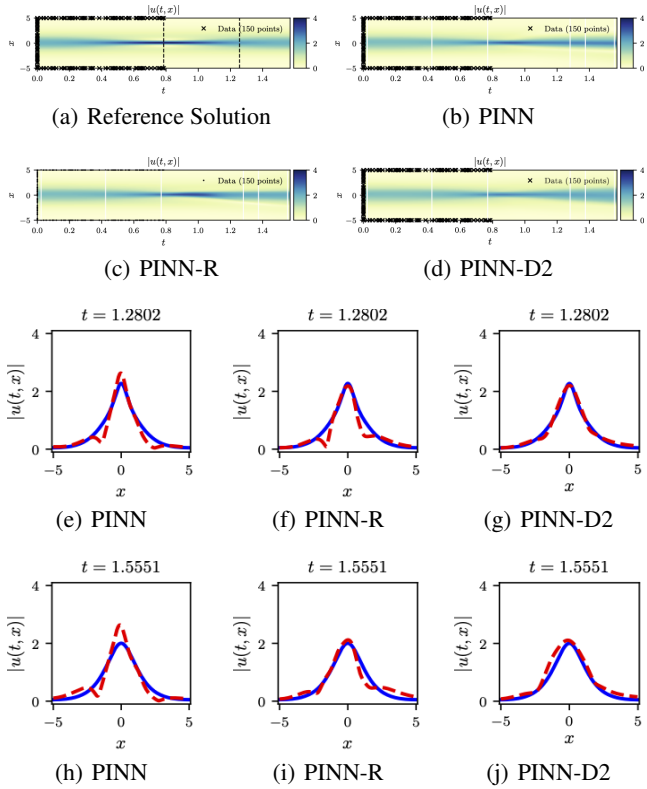


Figure 6: Top two rows: the complete reference solution and predictions of the nonlinear Schrödinger equation. The points marked with $\times$ mean initial or boundary points. Bottom: the extrapolation solution snapshots at $t = \{1.2802, 1.5551\}$.

4.3 Ablation Study

To show the efficacy of controlling δ in Eq. (9), we compare PINN-D1 and PINN-D2. In Table 1, PINN-D2 outperforms PINN-D1 for three benchmark equations. The biggest improvement is made in the NLS equation, one of the most difficult equations to predict, i.e., 0.314 vs. 0.141 in the L2-norm metric. We note that without controlling δ, PINN-D1 shows worse predictions even than PINN and PINN-R in this equation.

4.4 Visualization of Training Process

Fig. 7 shows the curves of L_u and L_f with our method in the benchmark viscous Burgers' equation. For L_f, we set $\epsilon = 0.001$, $\delta = 0.01$, and $w = 1.01$, which produces the best extrapolation accuracy. With this setting, DPM immediately pulls L_f toward the threshold $\epsilon = 0.001$ as soon as $L_f > 0.001$. Because our method uses the smallest manipulation vector, $\boldsymbol{v}^*$, L_u is also trained properly as training goes on.

4.5 PINN vs. Regression

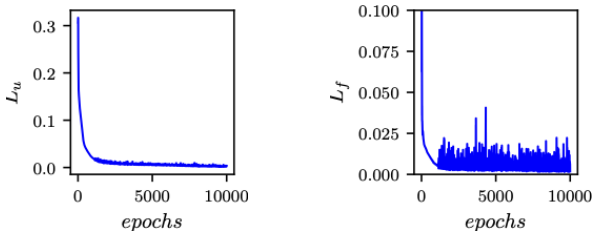
The task definition of PINN is different from that of the simple regression learning a solution function u, where the reference solutions of $u(x,t)$ are collected not only for initial and boundary conditions but also for other (x,t) pairs. In general, this approach requires non-trivial efforts to run computer simulations and collect such reference solutions. Once we collect them, one advantage is learning u becomes a simple regression task without involving L_f. However, a critical disadvantage is that governing equations cannot be explicitly imposed during the training process.

Although our task is not to fit a regression model to the reference solutions but to learn physical dynamics, we compare our proposed method with the regression-based approach to better understand our method. To train the regression model, we use L_u with an augmented training set $\left\{\left((x_u^i, t_u^i), u(x_u^i, t_u^i)\right)\right\}_{i=1}^{N_u} \cup \left\{\left((x_r^i, t_r^i), u(x_r^i, t_r^i)\right)\right\}_{i=1}^{N_r}$, where the first set consists of initial and boundary training samples, (x_r^i, t_r^i) are sampled uniformly in Ω and $[0, T_{\text{train}}]$, and we set $N_r = N_f$ for fairness. We run external software to calculate $u(x_r^i, t_r^i)$, which is not needed for $u(x_u^i, t_u^i)$ because initial and boundary conditions are known a priori.

We train two regression models: one based on a series of fully connected (FC) layers and the other based on residual connections. In Table 2, they are denoted by FC and FC-R, respectively. We note that the neural network architecture of FC (resp. FC-R) is the same as that of PINN (resp. PINN-R) but they are trained in the supervised manner described earlier. We use the same set of hyperparameters for the number of

PDE	L2-norm (↓)				Explained variance score (↑)				Max error (↓)				Mean absolute error (↓)			
	FC	FC-R	PINN-D1	PINN-D2	FC	FC-R	PINN-D1	PINN-D2	FC	FC-R	PINN-D1	PINN-D2	FC	FC-R	PINN-D1	PINN-D2
Vis. Burgers	0.352	0.301	0.112	**0.092**	0.896	0.915	0.988	**0.991**	0.718	0.598	0.545	**0.333**	0.119	0.108	0.026	**0.021**
Inv. Burgers	0.114	0.133	**0.083**	0.090	0.060	-0.181	0.454	**0.621**	3.245	3.301	**1.534**	2.036	0.255	0.332	**0.277**	0.315
Allen–Cahn	0.324	0.313	0.246	**0.182**	0.873	0.766	0.939	**0.967**	1.512	1.190	1.096	**0.8366**	0.207	0.336	0.129	**0.094**
Schrödinger	0.375	0.235	0.314	**0.141**	-3.438	-3.174	-4.973	**-3.257**	4.078	4.3165	4.945	**3.829**	2.072	1.868	**0.868**	0.896

Table 2: The extrapolation accuracy in terms of the relative errors in L2-norm, the Pearson correlation coefficient, and R^2 in various PDEs. Large (resp. small) values are preferred for ↑ (resp. ↓).



(a) L_u curve (b) L_f curve

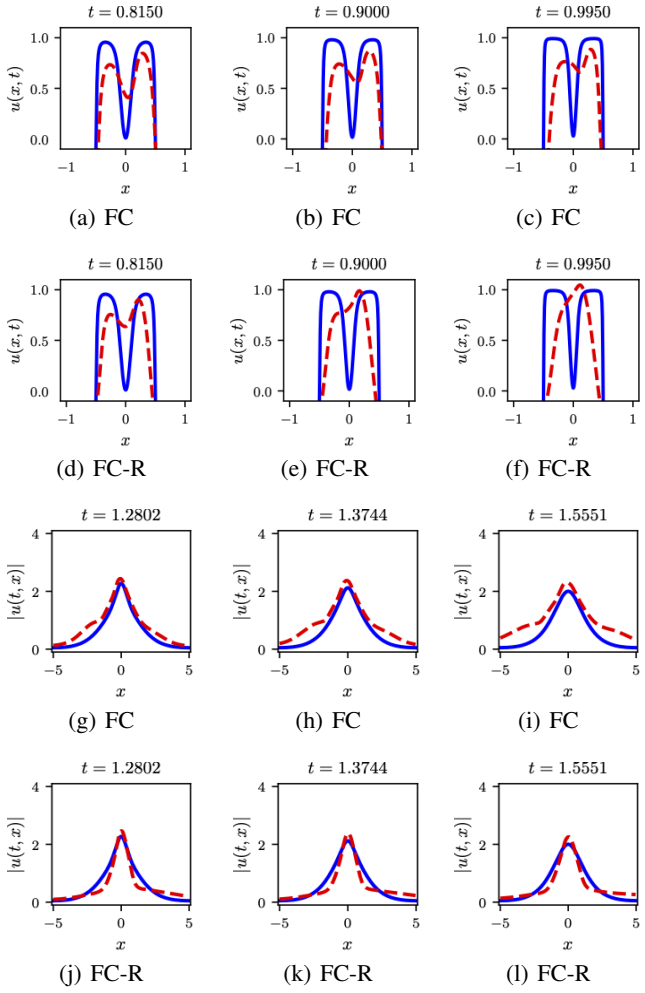
Figure 7: Example training curves of L_u and L_f of PINN-D2 ($\epsilon = 0.001, \delta = 0.01$, and $w = 1.01$) for the benchmark viscous Burgers' equation

layers, the dimensionality of hidden vector, and so on, and choose the best hyperparameter using the validation set. Note that this is exactly the same environment as the experiments shown in the main text.

Our proposed PINN-D1 and D2 outperform the regression-based models for all benchmark problems by large margins in Table 2. In Fig. 8, we show the extrapolation results of the two regression models for the worst and the best performing cases in terms of human visual perception. For the AC equation (the top row in the figure), it is hard to say that they learned the physical dynamics. In particular, FC-R shows the worst extrapolation in Fig. 8(f). On the other hand, FC-R is successful for the NLS equation (the bottom row in the figure), whereas FC fails to extrapolate the both ends of the curve. Therefore, we can say that the regression-based approach shows unstable performance and is not "physics-informed."

5 Conclusions

In this work, we presented a novel training method, dynamic pulling method (DPM), for obtaining better performing physics-informed neural networks in extrapolation. The proposed DPM enables PINN to learn dynamics of the governing equations accurately. In the numerical experiments, we first demonstrated that the original PINN performs poorly on extrapolation tasks and empirically analyzed PINN in detail. Then, we demonstrated that the proposed DPM significantly outperforms PINN and its residual-block variant (up to 72% in comparison with PINN and PINN-R) in various metrics. As an ablation study, we compared PINN-D1 and PINN-D2, where PINN-D2 overwhelms PINN-D1 in three benchmark problems. Finally, we explained how DPM behaves by illustrating example training loss curves. All codes and data will be released upon publication.



(a) FC (b) FC (c) FC
(d) FC-R (e) FC-R (f) FC-R
(g) FC (h) FC (i) FC
(j) FC-R (k) FC-R (l) FC-R

Figure 8: We visualize the results by regression models. As shown, they are inferior to our PINN-D2 in Figs. 5 and 6. Top two rows: the regression extrapolation snapshots for the Allen–Cahn equation. Bottom: the regression extrapolation snapshots for the nonlinear Schrödinger equation.

Acknowledgements

Noseong Park (noseong@yonsei.ac.kr) is the corresponding author. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University)). This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, a wholly owned subsidiary of Honeywell International, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

Anderson, D.; Tannehill, J. C.; and Pletcher, R. H. 2016. *Computational fluid mechanics and heat transfer*. Taylor & Francis.

Ben-Israel, A.; and Greville, T. 2006. *Generalized Inverses: Theory and Applications*. CMS Books in Mathematics. Springer New York. ISBN 9780387216348.

Cosmin Anitescu, Elena Atroshchenko, N. A. T. R. 2019. Artificial Neural Network Methods for the Solution of Second Order Boundary Value Problems. *Computers, Materials & Continua* 59(1).

Cranmer, M.; Greydanus, S.; Hoyer, S.; Battaglia, P.; Spergel, D.; and Ho, S. 2020. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630* .

Doan, N. A. K.; Polifke, W.; and Magri, L. 2019. Physics-Informed Echo State Networks for Chaotic Systems Forecasting. In Rodrigues, J. M. F.; Cardoso, P. J. S.; Monteiro, J.; Lam, R.; Krzhizhanovskaya, V. V.; Lees, M. H.; Dongarra, J. J.; and Sloot, P. M., eds., *Computational Science – ICCS 2019*, 192–198. Cham: Springer International Publishing.

Erichson, N. B.; Muehlebach, M.; and Mahoney, M. W. 2019. Physics-informed autoencoders for Lyapunov-stable fluid flow prediction. *arXiv preprint arXiv:1905.10866* .

Fulton, L.; Modi, V.; Duvenaud, D.; Levin, D. I.; and Jacobson, A. 2019. Latent-space Dynamics for Reduced Deformable Simulation. In *Computer graphics forum*, volume 38, 379–391. Wiley Online Library.

Geist, M.; Petersen, P.; Raslan, M.; Schneider, R.; and Kutyniok, G. 2020. Numerical Solution of the Parametric Diffusion Equation by Deep Neural Networks. *arXiv preprint arXiv:2004.12131* .

Greydanus, S.; Dzamba, M.; and Yosinski, J. 2019. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, 15353–15363.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.

Hirsch, C. 2007. *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*. Elsevier.

Holl, P.; Thuerey, N.; and Koltun, V. 2020. Learning to Control PDEs with Differentiable Physics. In *International Conference on Learning Representations*.

Hornik, K.; Stinchcombe, M.; White, H.; et al. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2(5): 359–366.

Iserles, A. 2009. *A first course in the numerical analysis of differential equations*. 44. Cambridge university press.

Kang, C.; Park, N.; Prakash, B. A.; Serra, E.; and Subrahmanian, V. S. 2016. Ensemble Models for Data-Driven Prediction of Malware Infections. In *WSDM*.

Khoo, Y.; Lu, J.; and Ying, L. 2017. Solving parametric PDE problems with artificial neural networks. *arXiv preprint arXiv:1707.03351* .

Kim, J.; Lee, K.; Lee, D.; Jin, S. Y.; and Park, N. 2020. DPM: A Novel Training Method for Physics-Informed Neural Networks in Extrapolation. *arXiv preprint arXiv:2012.02681* .

Kim, K. 2003. Financial time series forecasting using support vector machines. *Neurocomputing* 55(1): 307–319. Support Vector Machines.

Lee, K.; and Carlberg, K. 2019. Deep Conservation: A latent dynamics model for exact satisfaction of physical conservation laws. *arXiv preprint arXiv:1909.09754* .

Lee, K.; and Carlberg, K. T. 2020. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics* 404: 108973.

LeVeque, R. J.; et al. 2002. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press.

Ling, J.; Kurzawski, A.; and Templeton, J. 2016. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics* 807: 155–166.

Ling, J.; and Templeton, J. 2015. Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier–Stokes uncertainty. *Physics of Fluids* 27(8): 085103.

Raissi, M.; Perdikaris, P.; and Karniadakis, G. E. 2019. 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.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *Nature* 323(6088): 533–536.

Stoer, J.; and Bulirsch, R. 2013. *Introduction to numerical analysis*, volume 12. Springer Science & Business Media.

Tripathy, R. K.; and Bilionis, I. 2018. Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of computational physics* 375: 565–588.

Vlachas, P. R.; Byeon, W.; Wan, Z. Y.; Sapsis, T. P.; and Koumoutsakos, P. 2018. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474(2213): 20170844.

Wiewel, S.; Becher, M.; and Thuerey, N. 2019. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer Graphics Forum*, volume 38, 71–82. Wiley Online Library.

Yang, L.; Meng, X.; and Karniadakis, G. E. 2020. B-PINNs: Bayesian Physics-Informed Neural Networks for Forward and Inverse PDE Problems with Noisy Data. *ArXiv* abs/2003.06097.

Zhang, D.; Lu, L.; Guo, L.; and Karniadakis, G. E. 2019. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics* 397: 108850. ISSN 0021-9991.