Improved Training of Physics-Informed Neural Networks with Model Ensembles

Katsiaryna Haitsiukevich & Alexander Ilin
Aalto University
Espoo, Finland
{firstname.lastname}@aalto.fi

Abstract—Learning the solution of partial differential equations (PDEs) with a neural network is an attractive alternative to traditional solvers due to its elegance, greater flexibility and the ease of incorporating observed data. However, training such physics-informed neural networks (PINNs) is notoriously difficult in practice since PINNs often converge to wrong solutions. In this paper, we address this problem by training an ensemble of PINNs. Our approach is motivated by the observation that individual PINN models find similar solutions in the vicinity of points with targets (e.g., observed data or initial conditions) while their solutions may substantially differ farther away from such points. Therefore, we propose to use the ensemble agreement as the criterion for gradual expansion of the solution interval, that is including new points for computing the loss derived from differential equations. Due to the flexibility of the domain expansion, our algorithm can easily incorporate measurements in arbitrary locations. In contrast to the existing PINN algorithms with time-adaptive strategies, the proposed algorithm does not need a pre-defined schedule of interval expansion and it treats time and space equally. We experimentally show that the proposed algorithm can stabilize PINN training and yield performance competitive to the recent variants of PINNs trained with time adaptation.

Index Terms—Label propagation, Model ensembles, Partial differential equations, Physics-informed neural networks

I. INTRODUCTION

Partial differential equations (PDEs) are a powerful tool for modeling many real-world phenomena [1], [2]. When derived from the first principles, partial differential equations can serve as predictive models which do not require any data for tuning. When learned from data, they often outperform other models by incorporating the inductive bias of the continuity of the modeled domain (time or space) [3]–[5]. Inference in this type of models is done by solving partial differential equations, that is by finding a trajectory that satisfies the model equations and a set of initial and boundary conditions. Since analytic solutions exist only for a limited number of models, inference is typically done by numerical solvers of differential equations.

One way of solving differential equations is to approximate the solution by a neural network which is trained to satisfy a given set of differential equations, initial and boundary conditions. This approach is known in the literature under the name of physics-informed neural networks (PINNs) [6], [7] and it can be seen as a machine learning alternative to classical numerical solvers. Despite the conceptual simplicity and elegance of the method, training PINNs is notoriously difficult in practice [8], [9]. It requires balancing of multiple terms in the loss function [8], [10] and the commonly used neural network architectures and parameter initialization schemes may not work best for PINNs [11], [12].

It has been noted by many practitioners that training PINNs often results in convergence to bad solutions [9], [12], [13]. Several recent works [9], [14], [15] address this problem by splitting the time interval into sub-intervals and sequentially training PINNs on each sub-interval. This idea is often referred in the literature as time adaptation or time marching. The time-adaptive strategies make the training procedure of PINNs similar to classical numerical solvers which compute the solution gradually moving from the initial conditions towards the other end of the time interval. Similarly to classical solvers, many existing PINN algorithms are based on a pre-defined schedule of time adaptation.

In this work, we follow the idea of the gradual expansion of the solution interval when training PINNs. We propose to train an ensemble of PINNs and use the ensemble agreement (confidence) as the criterion for expanding the solution interval to new areas. Due to the flexibility of the domain expansion, in contrast to existing baselines, our algorithm can easily incorporate measurements in arbitrary locations: the algorithm will propagate the solution from all the locations where supervision targets are known (see Fig. 1 as an example). The proposed algorithm does not need a pre-defined schedule of interval expansion and it treats time and space equally. We experimentally show that the proposed algorithm can stabilize PINN training (see Fig. 2) and yield performance competitive to the recent variants of PINNs which use time adaptation.

II. BACKGROUND

A. Physics-informed neural networks

Physics-informed neural networks are neural networks which are trained to approximate the solution of a partial differential equation

\[
\frac{\partial u(x,t)}{\partial t} = f \left( \frac{\partial^2 u(x,t)}{\partial x^2}, \frac{\partial u(x,t)}{\partial x}, x, t \right)
\]

on the interval \( x \in [X_1, X_2], t \in [T_1, T_2] \) with initial conditions

\[
u(x, T_1) = u_0(x), \quad x \in [X_1, X_2]
\]
Fig. 1. The training procedure for the convection equation (9) – (10) with known solution for \( t = 0 \) and \( t = 2 \). Gradual expansion of the solution domain (the blue and black points) from both ends of the interval is possible due to the flexibility of the proposed approach. The dot color scheme is from Fig. 4.

Fig. 2. The violin plots of the relative error \( r \) for the proposed method (blue violins on the right) and strong PINN baselines in solving three systems (9) – (15). The proposed method yields stable training.

and boundary conditions

\[ g(u(x, t)) = 0, \quad x \in \{X_1, X_2\}. \quad (3) \]

Functions \( f \) and \( u_0 \) in (1)–(2) are assumed to be known, function \( g \) in (3) is known as well and it can represent different types of boundary conditions (e.g., Neumann, Dirichlet, Robin or periodic boundary conditions). In the PINN approach, one approximates the solution with a neural network which takes inputs \( x \) and \( t \) and produces \( u(x, t) \) as the output. The network is trained by minimizing a sum of several terms:

\[ \mathcal{L} = \mathcal{L}_S + \mathcal{L}_B + \mathcal{L}_{\text{PDE}}. \quad (4) \]

\( \mathcal{L}_S \) is the standard supervised learning loss which makes the neural network fit the initial conditions:

\[ \mathcal{L}_S = \sum_{i=1}^{K_S} w_i (u(x_i, t_i) - u_i)^2. \quad (5) \]

where \( w_i \) are point-specific weights, \( t_i = T_1, u_i = u_0(x_i) \) and \( x_i \) are sampled from \([X_1, X_2]\). \( \mathcal{L}_B \) is the loss computed to satisfy the boundary conditions:

\[ \mathcal{L}_B = \sum_{j=1}^{K_B} w_j ||g(u(x_j, t_j))||^2, \quad (6) \]

where \( w_j \) are point-specific weights, \( x_j \in \{X_1, X_2\} \) and \( t_j \) are sampled from \([T_1, T_2]\). \( \mathcal{L}_{\text{PDE}} \) is the loss derived from the PDE in (1):

\[ \mathcal{L}_{\text{PDE}} = \sum_{k=1}^{K_{\text{res}}} w_k \left( \frac{\partial^2 u_k}{\partial t^2} - f \left( \frac{\partial^2 u_k}{\partial x^2}, \frac{\partial u_k}{\partial x}, x_k, t_k \right) \right)^2, \quad (7) \]

where \( w_k \) are point-specific weights and the partial derivatives \( \frac{\partial u_k}{\partial t}, \frac{\partial^2 u_k}{\partial x^2}, \frac{\partial u_k}{\partial x} \) are computed at collocation points \((x_k, t_k)\) sampled from the interval \( x_k \in [X_1, X_2], t_k \in [T_1, T_2] \). Classical PINNs use shared weights \( w_S = w_i, \forall i, w_B = w_j, \forall j, w_{\text{PDE}} = w_k, \forall k \). The values of the weights vary depending on the implementation and in the simplest case they are \( w_S = 1/K_S, w_B = 1/K_B, w_{\text{PDE}} = 1/K_{\text{PDE}} \).

The method can easily be extended to fit a sequence of observations \( \{(x_i^*, t_i^*), w_i^*\}_i \) by including the observed data to the supervision loss (5). In this case, the method can be seen as fitting a neural network to the training data (containing the observations) while regularizing the solution using the PDE loss (7). One advantage of the PINN method compared to traditional numerical solvers is the ability to work with ill-posed problems, for example, if the initial conditions are known only in a subset of points.

B. PINN extensions

Despite the elegance of the PINN approach, the method is known to be prone to failures, especially when the solution has a complex shape on the considered interval (see, e.g., [8]–[11], [13], [14]). The optimization problem solved while training PINNs is very hard and the accuracy of the found solution is very sensitive to the hyperparameters of PINNs [16]: the weights \( w_S, w_B, w_{\text{PDE}} \) of the loss terms, the parameterization used in the neural network and the strategy for sampling the points in which the loss terms are computed.

Balancing shared weights \( w_S, w_B, w_{\text{PDE}} \) of the individual loss terms (5) – (7) is a popular way to improve the accuracy of the PINN solution. In [14] the authors propose using larger weights \( w_S \) relative to \( w_B \) and \( w_{\text{PDE}} \) because the initial conditions largely determine the shape of the solution. Several works propose different schemes for dynamically adapting \( w_S, w_B, w_{\text{PDE}} \) during training such that the weights that correspond to problematic loss terms get higher values. The weights can be adjusted either based on the magnitudes of the gradients of the corresponding loss terms [8] or based on the eigenvalues of the limiting Neural Tangent Kernel [10]. Alternatively, weights \( w_S, w_B, w_{\text{PDE}} \) can be treated as trainable parameters and adjusted jointly with the PINN parameters solving a minimax optimization problem [17]. Self-Adaptive PINNs (SA-PINNs) [18] increase point-wise weights \( w_i, w_j \) and \( w_k \) of the loss terms (5)–(7) during training such that the changes of the weights are proportional to the corresponding loss terms.

The accuracy of PINNs can also be improved by using a different parameterization for the trained neural network instead of the most standard multilayer perceptron architecture.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Several works [6], [19]–[21] propose neural network parameterizations which guarantee that the initial or boundary conditions are satisfied exactly, thus eliminating terms $\mathcal{L}_S$ and $\mathcal{L}_B$ from (4). In [11] Fourier features are used as the inputs of the neural network, which is motivated by the success of Fourier feature networks [22] in preserving high-frequencies in the modeled solution. The method called SIREN [12] proposes to use a multilayer perceptron with periodic activation functions and adjusts the weight initialization schemes to work better with such activation functions. This parameterization can also improve learning of high frequencies in the modeled solution. Multiplicative connections added to the standard multilayer perceptron architecture [8] help to account for possible multiplicative interactions between different input dimensions.

Another popular way of improving the accuracy of PINNs is to use adaptive strategies for sampling collocation points $(x_k, t_k)$ to compute $\mathcal{L}_{PDE}$ in (7). Sampling more collocation points in the areas with large values of $\mathcal{L}_{PDE}$ [14] helps to learn solutions with fast transitions. Daw et al. [23] propose an evolutionary strategy for sampling the collocation points: the points with large contribution to $\mathcal{L}_{PDE}$ are kept for the next iteration while the rest of the points are re-sampled uniformly from the domain. These approaches resemble the strategy of the classical solvers to reduce the discretization interval when the solution cannot be estimated accurately. More details on the sampling strategies for PINN and empirical comparison can be found in [24].

**C. PINNs with expansion of the time interval**

Recently, several papers have proposed to solve the initial-boundary value problem by gradually expanding the interval from which points $(x_k, t_k)$ in (7) are sampled. These versions of PINNs are closest to our approach.

**a) Time-adaptive strategies:** The time interval $[T_1, T_2]$ is split into multiple sub-intervals and the equation is solved on each individual sub-interval sequentially by a separate PINN [9], [14]. The solution at the border of the previous sub-interval is used as the initial condition for the next one. A similar approach is based on progressive expansion of the time interval by gradually increasing the end point $T_2$ during training [14], [15]. Backward compatible PINNs (bc-PINNs) [15] implement this idea such that the solution found during the previous interval extension is used as the PINN targets during training on a newly expanded interval to prevent catastrophic forgetting. All these time adaptation strategies need a pre-defined schedule for the time interval expansion, which makes them similar to classical numerical solvers which use pre-defined discretization schemes.

**b) Causality training:** The idea of time adaptation is closely related to the adaptive weighting scheme that respects causality [13]. The authors use adaptive weights for the individual terms in (7) such that the weights are computed using the cumulative PDE loss for the preceding points:

$$w_k = \exp \left( - \sum_{t_k'} |t_k' < t_k | \mathcal{L}_{PDE}(t_k') \right),$$

where $\mathcal{L}_{PDE}(t_k')$ denotes an individual term in (7) that corresponds to time point $t_k'$. The idea is to zero out the effect of the points that are far away from the initial conditions until the solution is approximated well on all the points before them. The method assumes that the boundary conditions are enforced as hard constraints and thus the total loss consists of $\mathcal{L}_S$ and $\mathcal{L}_{PDE}$.

### III. Ensembles of PINNs

**A. Motivation**

To motivate our approach, we demonstrate failure cases of PINNs using an example from [9] on solving a convection equation. Fig. 3 shows two inaccurate solutions (Figs. 3a, b) found by PINNs trained with the same network architecture but different random seeds for weight initialization. The found PINN solutions can be seen as a combination of two solutions: the correct solution near the initial conditions (for small $t$) and a simpler solution farther away from the initial conditions (for large $t$). The second row in Fig. 3 illustrates that the second, simpler solution satisfies well the solved PDEs. This example illustrates that simple solutions can be attractive for PINNs, which can cause problems for the optimization procedure. Once a PINN finds a simple, locally consistent solution in some areas (for example, far away from the initial conditions), it may be very difficult to change it. This leads to a final solution which is a combination of the correct solution and a wrong one.

This example provides the following intuition: when points located far away from the initial conditions contribute to the PDE loss (7), it may hurt the optimization procedure by pulling the solution towards a bad local optimum. On the other hand, including those points in the PDE loss at the beginning of training hardly brings any benefits: it makes little sense to regularize the solution using the loss (7) before we know its approximate shape. The ability to escape from wrong solutions
largely depends on the design choices made for the PINN training such as the optimizer type, the use of mini-batches, type of the sampling utilized for points in $\mathcal{L}_{\text{PDE}}$ term (7), normalization of the inputs, hard or soft encoding of the initial and boundary conditions and so on. While some of these tricks can be beneficial for the PINN accuracy on particular systems, it is quite difficult to select a common set of settings beneficial across a wider range of PDEs.

The illustrated problem is avoided by the classical numerical solvers because they usually “propagate” the solution from the initial and boundary conditions to cover the entire interval using a schedule determined by the discretization scheme.

B. Method

In this paper, we propose to gradually expand the areas from which we sample collocation points $(x_k, t_k)$ to compute loss $\mathcal{L}_{\text{PDE}}$. Our approach is based on training an ensemble of PINNs: a set of neural networks initialized with different weights but trained using the same loss function. Since PINN ensemble members typically converge to the same solution in the vicinity of observed data but may favor distinct wrong solutions farther away from the observations (Fig. 3a-b), we can use the ensemble agreement as the criterion for including new points for computing loss $\mathcal{L}_{\text{PDE}}$. Optionally, if all ensemble members agree on the solution in a particular point, we can create a pseudo-label (taken as the median of the ensemble predictions) for that point and make this point contribute to the supervision loss $\mathcal{L}_S$ in (5).

The proposed algorithm is illustrated in Fig. 1 and Fig. 4. At the beginning of training, the supervision loss $\mathcal{L}_S$ is computed using points with known solution (e.g., measurements, blue dots in Fig. 1) or points sampled at the initial conditions (blue dots in Fig. 4) and losses $\mathcal{L}_{\text{PDE}}$ and $\mathcal{L}_B$ are computed using only points that are close enough to the points with supervision targets (the black and red dots respectively). The proximity is measured by thresholding the Euclidean distance to the closest point among the blue dots. After $N$ training iterations, we compute the median and the variance of the ensemble predictions (see Fig. 4a). If the variance in a particular location is small enough, we use the median of the ensemble predictions at that point as a pseudo-label and add that point to the data set which is used to compute the supervision loss $\mathcal{L}_S$ (the light blue dots in Fig. 4a-b). Points for pseudo-labeling are selected among the collocation points (the black dots in Fig. 4). Then, we train the ensemble of PINNs for a fixed number of iterations and again increase the sets of inputs which are used to compute the losses (Fig. 3a-b). The iterations continue until all collocation points (which are pre-sampled at the beginning of the training procedure) are included in the loss computations.

More formally, the training procedure is presented in Algorithm 1. The dots and the circles in the algorithm refer to Fig. 4. In the experiments, we test two versions of the proposed algorithm:

1) **Pseudo-labels (PL):** a version with pseudo-labels in which the points with a high degree of ensemble agreement are used to compute both $\mathcal{L}_{\text{PDE}}$ and $\mathcal{L}_S$;

2) **PINN Ensemble (Ens):** a version without pseudo-labels,

---

**Algorithm 1 Training PINNs with label propagation**

Hyperparameters: $\Delta_{\text{PDE}}, \Delta, \sigma^2$ and $\epsilon$

1: $D_h = \{(x_i, t_i, u_i)\} \triangleright$ Points with targets (blue dots)
2: $D_{\text{PL}} = \{\}$ \triangleright Points with pseudo-labels (light blue dots)
3: $I_{\text{B}} = \{(x_j, t_j)\} \triangleright$ Points for $\mathcal{L}_B$ (red circles)
4: $I_{\text{PDE}} = \{(x_k, t_k)\} \triangleright$ Points for $\mathcal{L}_{\text{PDE}}$ (black circles)
5: while not converge do
6: $D = D_L \cup D_{\text{PL}}$ ('PL' version) or $D_L$ ('Ens' version), $I_{\text{B}}$ and $I_{\text{PDE}}$ to compute $\mathcal{L}_S$, $\mathcal{L}_B$, $\mathcal{L}_{\text{PDE}}$, respectively.
7: $I_{\text{B}} = \{(x_j, t_j) \in I_{\text{B}} \mid R((x_j, t_j), D) < \Delta_{\text{PDE}}\}$\triangleright red dots
8: $I_{\text{PDE}} = \{(x_k, t_k) \in I_{\text{PDE}} \mid R((x_k, t_k), D) < \Delta_{\text{PDE}}\}$\triangleright black dots
9: Train $L$ networks $f_i$ for $N$ iterations using $D_L \cup D_{\text{PL}}$ ('PL' version) or $D_L$ ('Ens' version), $I_{\text{B}}$ and $I_{\text{PDE}}$ to compute $\mathcal{L}_S$, $\mathcal{L}_B$, $\mathcal{L}_{\text{PDE}}$, respectively.
10: for $(x_k, t_k) \in I_{\text{PDE}}$ do
11: $\tilde{u}_i = f_i(x_k, t_k), \ \forall l \in 1, \ldots, L$
12: $u_k = \text{variance} (\tilde{u}_1, \ldots, \tilde{u}_L)$
13: $\bar{u}_k = \text{median} (\tilde{u}_1, \ldots, \tilde{u}_L)$
14: if $u_k < \sigma^2$ \& $R((x_k, t_k), D) < \Delta$ then
15: $D_{\text{PL}} \leftarrow D_{\text{PL}} \cup ((x_k, t_k), \bar{u}_k)$
16: end if
17: end for
18: end while
19:
20: function $R((x, t), D)$
21: $D' = \{(x_i, t_i) \in D \mid \|\frac{1}{L} \sum_i f_i(x_i, t_i) - u_i\| < \epsilon\}$ \triangleright Points with good fit to targets
22: return $\min_{(x_i, t_i) \in D'} \| (x_i, t_i) - (x, t) \|$
in which the points with a high degree of ensemble agreement are used to compute $\mathcal{L}_{\text{PDE}}$ but not $\mathcal{L}_S$.

Each member of the ensemble is trained to minimize the loss (4) with shared weights $w_B = 1/|I_B|$, $w_{\text{PDE}} = 1/|I_{\text{PDE}}|$ and $w_S = 1/|D|$ for the PINN Ensemble version and $w_S = 1/|D \cup I_{\text{PDE}}|$ for the version with pseudo-labels. Other weighting strategies (e.g., [8], [10], [14]) could be combined with our approach as well.

C. Related work

a) PINNs and their extensions: The proposed algorithm is built on the idea of the gradual expansion of the solution interval, which makes it similar to the time-adaptive techniques [9], [14], [15] and the adaptive weighting method that respects causality [13]. The advantage of the proposed algorithm is its greater flexibility in the way of expanding the area covered by collocation points: instead of expanding the time interval with a pre-defined [9], [14], [15] or an automatic [13] schedule, our algorithm considers each collocation point individually during the expansion and it treats time and space equally. This feature allows the application of the algorithm to datasets with an arbitrary layout of the points with known targets. We illustrate this by solving the convection system (9) – (10) on the interval $t \in [0, 2]$ when the solution is known for $t = 0$ and $t = 2$. As illustrated in Fig. 1, the algorithm finds a reasonable schedule for expanding the area starting from both ends of the interval.

b) Label propagation and ensembles: Training an ensemble of PINNs with pseudo-labeling is related to how label propagation is done in semi-supervised classification tasks [25], [26]: when a classifier becomes confident in the predicted class of an unlabeled example, that example is added to the labeled set. Model ensembles [27] or prediction ensembles [28] are often used in those tasks to generate better targets. Since PINNs are trained on real-valued targets, one can view PINNs as regression models regularized by the loss (7). Label propagation in regression tasks is less studied with only a few existing works on semi-supervised regression [29], [30]. In our algorithm, we use the confidence of the ensemble predictions to decide whether the solution interval can be extended and which points can be assigned pseudo-labels.

IV. EXPERIMENTS

We test the proposed algorithm on finding the solutions of the following differential equations on $x \in [0, 2\pi]$, $t \in [0, 1]$:

- convection equation used to model transport phenomena

$$\frac{\partial u}{\partial t} = -\beta \frac{\partial u}{\partial x}, \quad \beta = \text{const} \quad (9)$$

$$u(x, 0) = \sin(x), \quad u(0, t) = u(2\pi, t) \quad (10)$$

- reaction system for modelling chemical reactions

$$\frac{\partial u}{\partial t} = \rho u (1 - u), \quad \rho = \text{const} \quad (11)$$

$$u(x, 0) = \exp\left(-8(x - \pi)^2/\pi^2\right), \quad u(0, t) = u(2\pi, t) \quad (12)$$

- reaction-diffusion equation that models reactions together with diffusion of substances

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} + \rho u(1 - u), \quad \nu, \rho = \text{const} \quad (13)$$

$$u(x, 0) = \exp(-8(x - \pi)^2/\pi^2) \quad (14)$$

$$u(0, t) = u(2\pi, t), \quad u_x(0, t) = u_x(2\pi, t) \quad (15)$$

- diffusion equation with periodic boundary conditions

$$\frac{\partial u}{\partial t} = \frac{1}{d} \frac{\partial^2 u}{\partial x^2}, \quad d = \text{const} \quad (16)$$

$$u(x, 0) = \sin(dx) \quad (17)$$

$$u(0, t) = u(2\pi, t), \quad u_x(0, t) = u_x(2\pi, t) \quad (18)$$

- diffusion (16) with Dirichlet boundary conditions:

$$u(0, t) = u(2\pi, t) = 0 \quad (19)$$

In all the experiments, we use a multi-layer perceptron with four hidden layers with 50 neurons and the tanh activation in each hidden layer as a backbone PINN. Our ensemble of PINNs contains five such networks. The two inputs $x$ and $t$ of the network are normalized to $[-1, 1]$, which has a positive effect on the accuracy in our experiments. For all comparison methods (similarly to [9]), we use $K_{\text{PDE}} = 1000$ collocation points randomly sampled from a regular $256 \times 100$ grid on the solution interval. To fit the initial and boundary conditions, $K_S = 256$ and $K_B = 100$ points are selected uniformly from the corresponding domain boundaries. The models are trained either with the Adam optimizer [31] with learning rate 0.001 or with Adam followed by fine-tuning with limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (LBFGS) [32]. The LBFGS fine-tuning is done before each update of the collocation points which contribute to loss $\mathcal{L}_{\text{PDE}}$ and at the very end of training. We also set weighting coefficient $w_s = 64/|D \cup I_{\text{PDE}}|$ for PL version trained with Adam and LBFGS similar to the closest baselines.

Illustrations of the training procedure for the considered systems can be found in Figs. 4 and 5. The plots show that the proposed algorithm finds accurate solutions.

To evaluate the accuracy of the proposed algorithm, we report the relative $l_2$ error (from the previous works [9], [13])

$$r = l_2(\hat{u} - u)/l_2(u) \quad (20)$$

where $\hat{u}$ is a vector of the ground-truth values in the test set, $u$ is the corresponding PINN predictions and $l_2(\cdot)$ denotes $l_2$ norm. Our test set consists of points on a regular $256 \times 100$ grid.

In Table I, we present the means and standard deviations of the solution errors obtained in 10 runs with different initializations. We compare the accuracy of our method with several strong baselines proposed recently in the literature (with listed adjustments to the authors’ implementations):

1. Vanilla PINN trained with the LBFGS optimizer
2. Vanilla PINN trained with the Adam optimizer

The source code is available at https://github.com/katehai/ens_pinn.
3) **Causality [13]:** PINN with an adaptive weighting scheme that respects causality. We report results for the values of hyperparameter $\epsilon$ in (8) that worked best for the considered systems: $\epsilon = 0.01$ for convection, $\epsilon = 100$ for reaction and $\epsilon = 1$ for reaction-diffusion. We train the model until all the adapted weights become greater than 0.99 (the stopping criterion used in [13]). As the existing implementation requires a regular grid, loss $L_{pde}$ is computed for $K_{pde} = 32 \times 32 = 1024$ and $K_{pde} = 256 \times 100 = 25600$ collocation points.

4) **SA-PINN [18]:** with trainable point-specific weights. We switch off finetuning with the LBFGS optimizer after Adam due to observed training instabilities. We also add normalization of network inputs to $[-1, 1]$ similar to our method.

5) **bc-PINN [15]:** a technique of expanding the time interval with a pre-defined schedule. We expand the time interval four times and sample 250 collocation points on each interval to compute loss $L_{pde}$ and 25 points per interval to compute $L_B$.

The results in Table I show that vanilla PINNs trained with LBFGS struggle to find accurate solutions for all the considered systems. Vanilla PINNs trained with Adam yield more accurate results but the results are unsatisfactory for the reaction and reaction-diffusion systems. The causality weighting scheme seems to require many more collocation points to find satisfactory solutions. SA-PINN and bc-PINN work very well on the reaction and reaction-diffusion systems but do not find good solutions for the convection system. The proposed ensemble method provides stable training (see the summary of the results in Fig. 2) and shows competitive performance in all the considered systems. We can also observe that the LBFGS fine-tuning has positive effect on the accuracy when a good approximation is found during pre-training with Adam.

Next we compare the proposed algorithm on solving the diffusion system with the two types of boundary conditions: periodic (18) and Dirichlet-type (19). In this comparison, we omit the causality-motivated baseline [13] because the periodic boundary conditions in the existing implementations...
are enforced as hard constraints. The results in Table II show that training ensembles of PINNs yields competitive performance in these settings as well.

In Table III, we study the sensitivity of the ensemble training of PINNs to its hyperparameters. The results show that the PINN Ensemble algorithm is generally stable at solving the considered systems with little sensitivity to the hyperparameter values. We note the importance of hyperparameter consideration systems with little sensitivity to the hyperparameter PINN Ensemble algorithm is generally stable at solving the results in Table II show that training ensembles of PINNs yields competitive performance after a fixed number of updates. Those runs are excluded from the statistics.

Table III 'PINN Ensemble' algorithm results for different hyperparameter values. We report the same metric and systems as in Table I. The superscript (v) indicates how many runs out of 10 have less than 95% of the collocation points added to \( \mathcal{L}_{\text{PDE}} \) loss computation after a fixed number of updates. Those runs are excluded from the statistics.

| Ablation group, default value | Changed value | Convection, \( \beta \) | Reaction, \( \rho \) | Reaction-diffusion, \( \nu \) |
|-----------------------------|---------------|----------------|----------------|----------------|
| ensemble disagreement threshold, \( \sigma^2 = 4 \cdot 10^{-4} \) | \( \sigma^2 = 10^{-4} \) | 8.15 (2.7) (1) | 14.0 (2) | 7.16 (1.6) | 11.3 (2) | 12.3 (3) (3) |
| ensemble disagreement threshold, \( \sigma^2 = 10^{-3} \) | 8.11 (2.0) | 14.5 (4) | 7.16 (1.5) | 11.3 (2) | 13.0 (4) |
| distances for candidate collocation points, \( \Delta_{\text{PDE}} = 0.1, \Delta = 0.05 \) | \( \Delta_{\text{PDE}} = 0.07 \) | 10.1 (4.4) | 18 (10) (2) | 6.80 (1.1) | 11.5 (3) | 10.7 (3) (3) |
| distances for candidate collocation points, \( \Delta_{\text{PDE}} = 0.25 \) | 5.33 (9.9) | 11.5 (2) | 7.61 (2) | 12.7 (3) | 11.6 (4) |
| distances for candidate collocation points, \( \Delta_{\text{PDE}} = 0.125 \) | 7.82 (1.6) | 13.2 (2) | 7.67 (1.5) | 12.0 (2) | 15.3 (4) |
| distances for candidate collocation points, \( \Delta_{\text{PDE}} = 10^{3} \) | 6.25 (1.6) | 13.0 (2) | 8.42 (2) | 14.2 (3) | 14.5 (8) |
| distances for candidate collocation points, \( \Delta_{\text{PDE}} = 10^{5} \) | 8.98 (1.9) | 13.7 (2) | 8.33 (1.6) | 13.6 (2) | 18.4 (3) |
| distances for candidate collocation points, \( \Delta_{\text{PDE}} = 10^{7} \) | 11.1 (1.2) | 16.9 (3) | 8.22 (1.5) | 15.6 (7) | 36.3 (53) |
| prediction error for candidate collocation points, \( \epsilon = 10^{-3} \) | \( \epsilon = 5 \cdot 10^{-4} \) | 8.94 (2.3) | 13.9 (6) | 7.13 (1.5) | 11.2 (2) | 12.9 (4) (1) |
| prediction error for candidate collocation points, \( \epsilon = 10^{-2} \) | 7.65 (1.7) | 12.3 (2) | 7.15 (1.5) | 11.4 (2) | 13.0 (4) (1) |
| prediction error for candidate collocation points, \( \epsilon = 10^{-3} \) | 7.19 (1.4) | 12.2 (2) | 7.11 (1.5) | 11.4 (2) | 12.9 (4) (1) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{3} \) | \( \mathcal{K}_{\text{PDE}} = 5000 \) | 8.02 (1.9) | 14.1 (2) | 6.75 (1.4) | 10.7 (2) | 10.4 (3) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{4} \) | 7.71 (1.7) | 16.8 (4) | 7.00 (1.3) | 9.89 (2) | 10.3 (4) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{5} \) | \( \mathcal{K}_{\text{PDE}} = 50000 \) | 7.71 (1.7) | 16.8 (4) | 6.75 (1.4) | 10.7 (2) | 10.4 (3) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{4} \) | 7.71 (1.7) | 16.8 (4) | 7.00 (1.3) | 9.89 (2) | 10.3 (4) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{5} \) | \( \mathcal{K}_{\text{PDE}} = 500000 \) | 7.71 (1.7) | 16.8 (4) | 6.75 (1.4) | 10.7 (2) | 10.4 (3) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{4} \) | 7.71 (1.7) | 16.8 (4) | 7.00 (1.3) | 9.89 (2) | 10.3 (4) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{5} \) | \( \mathcal{K}_{\text{PDE}} = 5000000 \) | 7.71 (1.7) | 16.8 (4) | 6.75 (1.4) | 10.7 (2) | 10.4 (3) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{7} \) | 8.94 (1.9) | 14.1 (2) | 6.75 (1.4) | 10.7 (2) | 10.4 (3) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{7} \) | 8.02 (1.9) | 14.1 (2) | 6.75 (1.4) | 10.7 (2) | 10.4 (3) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{7} \) | 7.71 (1.7) | 16.8 (4) | 7.00 (1.3) | 9.89 (2) | 10.3 (4) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{7} \) | \( \mathcal{K}_{\text{PDE}} = 50000000 \) | 7.71 (1.7) | 16.8 (4) | 6.75 (1.4) | 10.7 (2) | 10.4 (3) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{7} \) | 7.71 (1.7) | 16.8 (4) | 7.00 (1.3) | 9.89 (2) | 10.3 (4) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{7} \) | \( \mathcal{K}_{\text{PDE}} = 500000000 \) | 7.71 (1.7) | 16.8 (4) | 6.75 (1.4) | 10.7 (2) | 10.4 (3) |
| number of collocation points, \( \mathcal{K}_{\text{PDE}} = 10^{7} \) | 7.71 (1.7) | 16.8 (4) | 7.00 (1.3) | 9.89 (2) | 10.3 (4) |

which may negatively affect the ensemble diversity. Other important hyperparameter is variance threshold for ensemble disagreement \( \sigma^2 \). When \( \sigma^2 \) is small then the interval expansion slows down while too large values of \( \sigma^2 \) can result in adding new regions in the training procedure too early.

V. DISCUSSION AND FUTURE WORK

In this paper, we propose to stabilize training of PINNs by gradual expansion of the solution interval based on the agreement of an ensemble of PINNs. The obtained results suggest that the proposed approach can reduce the number of failure cases during PINN training. Another potential advantage of the proposed algorithm is that the PINN ensemble produces confidence intervals which can be viewed as uncertainty estimates of the found solution (see the bottom row in Fig. 4).
Although the proposed algorithm is more computationally expensive compared to vanilla PINNs, ensemble training can be effectively parallelized in which case the wall clock time of training does not grow significantly. The method shows good results for tested systems.

This work can be extended in a number of ways. One potential direction is to use different ways of creating model ensembles, for example, by dropout [35] or pseudo-ensembles [36]. It is interesting to investigate how the proposed algorithm can be combined with other tricks from the PINN literature, for example, adaptive balancing of the loss terms [10]. Another direction is to find alternative ways of the solution interval expansion (e.g., update sets \( T^i_{\text{PDE}} \) more frequently), which may increase the convergence speed. It should also be possible to improve the PINN architecture such that the knowledge of a found (local) solution in one area could be used in finding the solution in another area. Using neural networks with the right inductive bias (e.g., similar to the ones proposed by [5], [37], [38]), might provide a solution for that.

REFERENCES

[1] L. C. Evans, Partial Differential Equations, ser. Graduate studies in mathematics. American Mathematical Society, 1998, vol. 19.

[2] R. Courant and D. Hilbert, Methods of mathematical physics: partial differential equations. John Wiley & Sons, Ltd, 1989.

[3] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, “Neural ordinary differential equations,” Advances in Neural Information Processing Systems, vol. 31, 2018.

[4] Y. Rubanova, R. T. Chen, and D. K. Duvenaud, “Latent ordinary differential equations for irregularly-sampled time series,” Advances in Neural Information Processing Systems, vol. 32, 2019.

[5] V. Iakovenko, M. Heinonen, and L. Lähdesmäki, “Learning continuous-time PDEs from sparse data with graph neural networks,” International Conference on Learning Representations, 2021.

[6] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural networks for solving ordinary and partial differential equations,” IEEE transactions on neural networks, vol. 9, no. 5, pp. 987–1000, 1998.

[7] 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, vol. 378, pp. 686–707, 2019.

[8] S. Wang, Y. Teng, and P. Perdikaris, “Understanding and mitigating gradient blowup pathways in physics-informed neural networks,” SIAM Journal on Scientific Computing, vol. 43, no. 5, pp. A3055–A3081, 2021.

[9] A. S. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney, “Characterizing possible failure modes in physics-informed neural networks,” Advances in Neural Information Processing Systems, vol. 34, 2021.

[10] S. Wang, X. Yu, and P. Perdikaris, “When and why PINNs fail to train: A neural tangent kernel perspective,” Journal of Computational Physics, p. 110768, 2021.

[11] S. Wang, H. Wang, and P. Perdikaris, “On the eigenvector bias of fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks,” Computer Methods in Applied Mechanics and Engineering, vol. 384, p. 113938, 2021.

[12] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, “Implicit neural representations with periodic activation functions,” in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 7462–7473.

[13] S. Wang, S. Sankaran, and P. Perdikaris, “Respecting causality is all you need for training physics-informed neural networks,” arXiv preprint arXiv:2203.07404, 2022.

[14] C. L. Wight and J. Zhao, “Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks,” arXiv preprint arXiv:2007.04542, 2020.

[15] R. Mattey and S. Ghosh, “A novel sequential method to train physics informed neural networks for Allen-Cahn and Cahn-Hilliard equations,” Computer Methods in Applied Mechanics and Engineering, vol. 390, p. 114474, 2022.

[16] S. Markidis, “The old and the new: Can physics-informed deep-learning replace traditional linear solvers?” Frontiers in Big Data, p. 92, 2021.

[17] D. Liu and Y. Wang, “A dual-dimer method for training physics-constrained neural networks with minmax architecture,” Neural Networks, vol. 136, pp. 112–125, 2021.

[18] L. McClenney and U. Braga-Neto, “Self-adaptive physics-informed neural networks using a soft attention mechanism,” arXiv preprint arXiv:2009.04544, 2020.

[19] S. Dong and N. Ni, “A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks,” Journal of Computational Physics, vol. 435, p. 110242, 2021.

[20] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, “Physics-informed neural networks with hard constraints for inverse design,” SIAM Journal on Scientific Computing, vol. 43, no. 6, pp. B1105–B1132, 2021.

[21] N. Sukumar and A. Srivastava, “Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks,” Computer Methods in Applied Mechanics and Engineering, vol. 389, p. 114333, 2022.

[22] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, “Fourier features let networks learn high frequency functions in low dimensional domains,” in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 7537–7547.

[23] A. Daw, J. Bu, S. Wang, P. Perdikaris, and A. Karpatne, “Rethinking the importance of sampling in physics-informed neural networks,” arXiv preprint arXiv:2207.02338, 2022.

[24] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, “A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks,” arXiv preprint arXiv:2207.10289, 2022.

[25] D.-H. Lee et al., “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks,” in Workshop on challenges in representation learning, ICML, vol. 3, 2013, p. 896.

[26] K. Sohn, D. Berthelot, N. Carlini, Z. Zhang, H. Zhang, C. A. Raffel, E. D. Cubuk, A. Kurakin, and C.-L. Li, “Fixmatch: Simplifying semi-supervised learning with consistency and confidence,” Advances in Neural Information Processing Systems, vol. 33, pp. 596–608, 2020.

[27] S. Laine and T. Aila, “Temporal ensembling for semi-supervised learning,” International Conference on Learning Representations, 2017.

[28] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. A. Raffel, “Mixmatch: A holistic approach to semi-supervised learning,” Advances in Neural Information Processing Systems, vol. 32, 2019.

[29] N. Jean, S. M. Xie, and S. Ermon, “Semi-supervised deep kernel learning: Regression with unlabeled data by minimizing predictive variance,” Neural Information Processing Systems, 2018.

[30] Y.-F. Li, H.-W. Zha, and Z.-H. Zhou, “Learning safe prediction for semi-supervised regression,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31, no. 1, Feb. 2017.

[31] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.

[32] D. C. Liu and J. Nocedal, “On the limited memory BFGS method for large scale optimization,” Mathematical programming, vol. 45, no. 1, pp. 503–528, 1990.

[33] J. C. Wong, C. Ooi, A. Gupta, and Y.-S. Ong, “Learning in sinusoidal spaces with physics-informed neural networks,” IEEE Transactions on Artificial Intelligence, 2022.

[34] F. M. Rohrhofer, S. Posch, C. Gößnitzer, and B. C. Geiger, “Understanding the difficulty of training physics-informed neural networks on dynamical systems,” arXiv preprint arXiv:2203.13648, 2022.

[35] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” arXiv preprint arXiv:1207.0580, 2012.

[36] P. Bachman, O. Alsharif, and D. Precup, “Learning with pseudo-ensembles,” Advances in Neural Information Processing Systems, vol. 27, 2014.

[37] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, “Learning to simulate complex physics with graph networks,” in Int. Conf. on Mach. Lear. PMLR, 2020, pp. 8459–8468.

[38] J. Brandstetter, D. E. Worrall, and M. Welling, “Message passing neural PDE solvers,” Int. Conf. on Learn. Represent., 2022.