Improved Training of Physics-Informed Neural Networks with Model Ensembles

Anonymous authors
Paper under double-blind review

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.

1 Introduction

Partial differential equations (PDEs) are a powerful tool for modeling many real-world phenomena (Evans, 1998; Courant & Hilbert, 1989). 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) (Chen et al., 2018; Rubanova et al., 2019; Iakovlev et al., 2021). 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 (most likely derived from the first principles), 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, Lagaris et al., 1998; Raissi et al., 2019) 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 (Wang et al., 2021a; Krishnapriyan et al., 2021). It requires balancing of multiple terms in the loss function (Wang et al., 2021a;c) and the commonly used neural network architectures and parameter initialization schemes may not work best for PINNs (Wang et al., 2021b; Sitzmann et al., 2020).

It has been noted by many practitioners that training PINNs often results in convergence to bad solutions (see, e.g., Krishnapriyan et al., 2021; Sitzmann et al., 2020; Wang et al., 2022). Several recent works (Wight & Zhao, 2020; Krishnapriyan et al., 2021; Mattey & Ghosh, 2022) 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

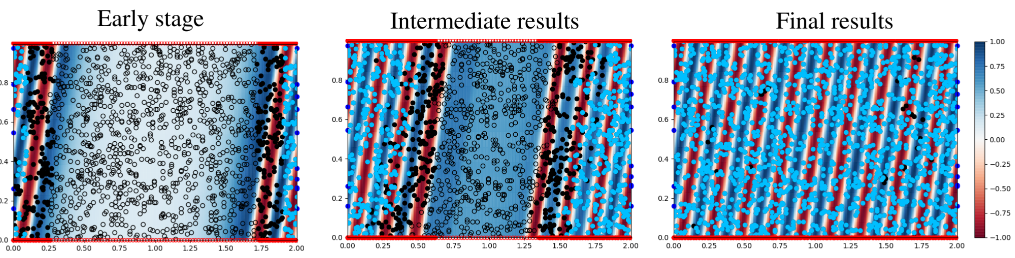


Figure 1: The training procedure for the convection equation (Eqs. 9– 10) when the solution is known for $t = 0$ and $t = 2$. Gradual expansion of the solution domain (represented by the blue and black points) from both ends of the interval is possible due to the flexibility of the proposed approach. The dot color scheme is from Fig. 4.

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.

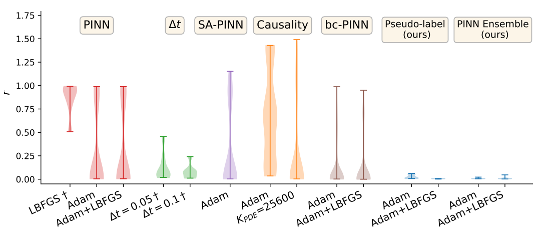


Figure 2: The violin plots of the relative error r for the proposed method (blue violins on the right) and strong PINN baselines in solving three systems (Eqs. 9–14). The proposed method yields stable training.

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.

2 BACKGROUND

2.1 PHYSICS-INFORMED NEURAL NETWORKS

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

$$\frac{\partial u(x,t)}{\partial t} = f\left(\frac{\partial^2 u(x,t)}{\partial x^2}, \frac{\partial u(x,t)}{\partial x}, x, t\right) \tag{1}$$

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

$$u(x, T_1) = u_0(x), \quad x \in [X_1, X_2] \tag{2}$$

and boundary conditions

$$\boldsymbol{g}(u(x,t)) = 0, \quad x \in \{X_1, X_2\}\,. \tag{3}$$

Functions f and u_0 in Eqs. 1–2 are assumed to be known, function $\boldsymbol{g}$ in Eq. 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:

$$\mathcal{L} = \mathcal{L}_{\text{S}} + \mathcal{L}_{\text{B}} + \mathcal{L}_{\text{PDE}}. \tag{4}$$

$\mathcal{L}_\text{S}$ is the standard supervised learning loss which makes the neural network fit the initial conditions:

$$\mathcal{L}_\text{S} = \sum_{i=1}^{K_\text{S}} w_i \left(u(x_i, t_i) - u_i\right)^2 . \tag{5}$$

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

$$\mathcal{L}_\text{B} = \sum_{j=1}^{K_\text{B}} w_j ||\boldsymbol{g}(u(x_j, t_j))||^2 , \tag{6}$$

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:

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

where w_k are point-specific weights and the partial derivatives $\frac{du_k}{dt}$, $\frac{\partial^2 u_k}{\partial x^2}$, $\frac{\partial u_k}{\partial x}$ 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 $w_\text{S} = w_i, \forall i$, $w_\text{B} = w_j, \forall j$, $w_\text{PDE} = w_k, \forall k$. The values of the weights vary depending on the implementation and in the simplest case they are $w_\text{S} = 1/K_\text{S}$, $w_\text{B} = 1/K_\text{B}$, $w_\text{PDE} = 1/K_\text{PDE}$.

The method can easily be extended to fit a sequence of observations $\{((x_i^*, t_i^*), u_i^*)\}_{i=1}^N$ by including the observed data to the supervision loss in Eq. 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 in Eq. 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. More details on existing extensions of PINNs can be found in Appendix A.1.

2.2 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 Eq. 7 are sampled. These versions of PINNs are closest to our approach.

Time-adaptive strategies Wight & Zhao (2020) and Krishnapriyan et al. (2021) show the effectiveness of the time marching strategy: 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. The solution at the border of the previous sub-interval is used as the initial condition for the next one.

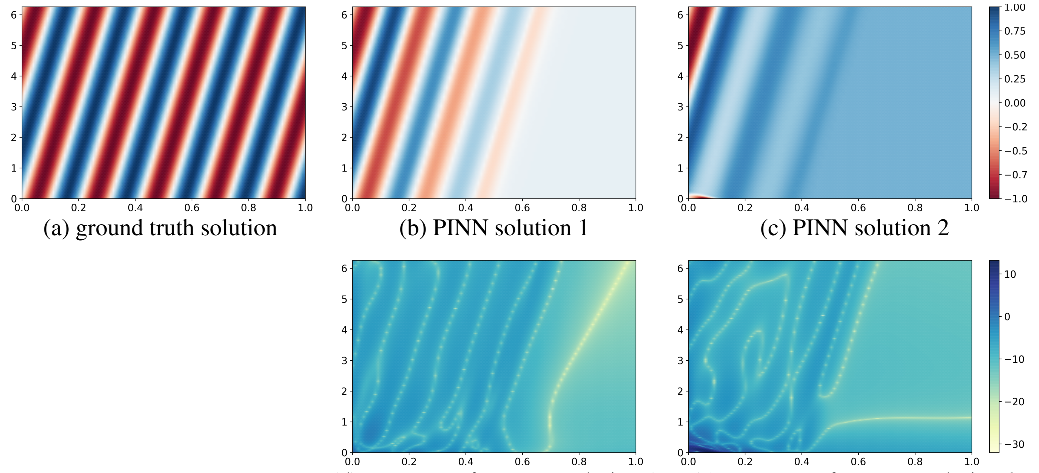
A similar approach is based on progressive expansion of the time interval by gradually increasing the end point T_2 during training (Wight & Zhao, 2020; Mattey & Ghosh, 2022). Backward compatible PINNs (bc-PINNs, Mattey & Ghosh, 2022) implement this idea such that the solution found during the previous 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.

Causality training The idea of time adaptation is closely related to the adaptive weighting scheme that respects causality (Wang et al., 2022). The authors use adaptive weights for the individual terms in Eq. 7 such that the weights are computed using the cumulative PDE loss for the preceding points:

$$w_k = \exp\left(- \textstyle\sum_{k'|t_{k'}<t_k} \epsilon \mathcal{L}_\text{PDE}(t_{k'})\right) , \tag{8}$$

where $\mathcal{L}_\text{PDE}(t_{k'})$ denotes an individual term in Eq. 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 $\mathcal{L}_\text{S}$ and $\mathcal{L}_\text{PDE}$.



(a) ground truth solution (b) PINN solution 1 (c) PINN solution 2

(d) PDE error for PINN solution 1 (e) PDE error for PINN solution 2

Figure 3: Example of solving the convection system (Eqs. 9–10) for $\beta = 30$, $t \in [0, 1]$ using PINNs with the LFBGS optimizer. (a): The ground truth solution. (b)-(c): Two solutions found with different initializations of PINNs. (d)-(e): The logarithm of the individual terms in Eq. 7 as a function of x and t for the solutions.

3 Ensembles of PINNs

3.1 Motivation

To motivate our approach, we demonstrate failure cases of PINNs using an example from (Krishnapriyan et al., 2021) on solving a convection equation. Fig. 3 shows the ground-truth solution of the equation (Fig. 3a) and two inaccurate solutions (Figs. 3b, c) 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 in Eq. 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 in Eq. 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 in Eq. 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.

3.2 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. 3b-c), 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 $\mathcal{L}_{\text{S}}$ in Eq. 5.

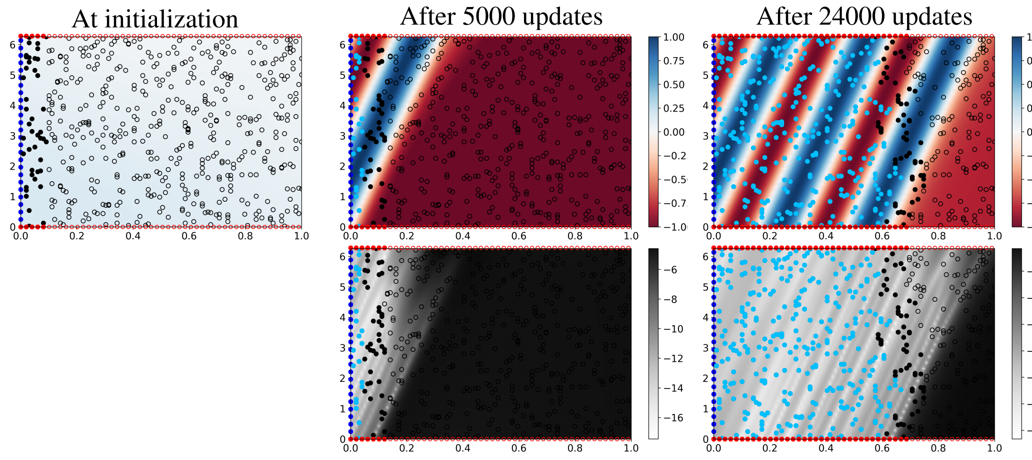


Figure 4: Illustration of the proposed algorithm on solving the convection system with $\beta = 20$ on $t \in [0, 1]$. Top row: ensemble median, bottom row: log-variance of ensemble predictions. Dark blue dots: set D_{L} of points which contribute to loss $\mathcal{L}_{\text{S}}$; light blue dots: set D_{PL} of points with pseudo-labels (contribute either to $\mathcal{L}_{\text{S}}$ or $\mathcal{L}_{\text{PDE}}$); black dots: points that contribute to loss $\mathcal{L}_{\text{PDE}}$; red dots: points that contribute to loss $\mathcal{L}_{\text{B}}$.

Algorithm 1 Training PINNs with label propagation

```
Hyperparameters: $\Delta_{\text{PDE}}$, $\Delta$, $\sigma^2$ and $\epsilon$
1:  $D_{\text{L}} = \{((x_i, t_i), u_i)\}$    ▷ Points with targets (blue dots)
2:  $D_{\text{PL}} = \{\}$    ▷ Points with pseudo-labels (light blue dots)
3:  $I_{\text{B}} = \{(x_j, t_j)\}$    ▷ Sample candidate points for computing $\mathcal{L}_{\text{B}}$ (empty red circles)
4:  $I_{\text{PDE}} = \{(x_m, t_m)\}$    ▷ Sample candidate points for computing $\mathcal{L}_{\text{PDE}}$ (empty black circles)
5:  while not converge do
6:      $D = D_{\text{L}} \cup D_{\text{PL}}$
7:      $I'_{\text{B}} = \{(x_j, t_j) \in I_{\text{B}} \mid \text{DISTANCE}((x_j, t_j), D) < \Delta_{\text{PDE}}\}$    ▷ red dots
8:      $I'_{\text{PDE}} = \{(x_m, t_m) \in I_{\text{PDE}} \mid \text{DISTANCE}((x_m, t_m), D) < \Delta_{\text{PDE}}\}$    ▷ black dots
9:      Train $L$ networks $f_l$ for $N$ iterations using $D_{\text{L}} \cup D_{\text{PL}}$ ('PL' version) or $D_{\text{L}}$ ('Ens' version),
        $I'_{\text{B}}$, $I'_{\text{PDE}}$ to compute $\mathcal{L}_{\text{S}}$, $\mathcal{L}_{\text{B}}$, $\mathcal{L}_{\text{PDE}}$, respectively.
10:     for $(x_m, t_m) \in I'_{\text{PDE}}$ do
11:         $\hat{u}_l = f_l(x_m, t_m), \quad \forall l \in 1, ..., L$    ▷ Prediction of each ensemble network
12:         $v_m = \text{variance}(\hat{u}_1, \dots, \hat{u}_L)$    ▷ Variance of predictions
13:         $\bar{u}_m = \text{median}(\hat{u}_1, \dots, \hat{u}_L)$    ▷ Median of predictions
14:         if $v_m < \sigma^2$ & $\text{DISTANCE}((x_m, t_m), D) < \Delta$ then
15:             $D_{\text{PL}} \leftarrow D_{\text{PL}} \cup ((x_m, t_m), \bar{u}_m)$    ▷ Add a point with a pseudo-label
16:         end if
17:     end for
18: end while
19:
20: function DISTANCE($(x, t)$, $D$)
21:     $D' = \{(x_i, t_i) \in D \mid ||\frac{1}{L}\sum_l f_l(x_i, t_i) - u_i|| < \epsilon\}$    ▷ Points with good fit to targets
22:     return $\min_{(x_i, t_i) \in D'} ||(x_i, t_i) - (x, t)||$
23: end function
```

The proposed algorithm is illustrated in Fig. 4. At the beginning of training, the supervision loss $\mathcal{L}_{\text{S}}$ is computed using points sampled at the initial conditions (the blue dots in Fig. 4a) and losses $\mathcal{L}_{\text{PDE}}$

and $\mathcal{L}_{\text{B}}$ are computed using only points that are close enough to the initial conditions (the black and red dots in Fig. 4a 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. 4b-c). 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 $\mathcal{L}_{\text{S}}$ (the light blue dots in Fig. 4b). Points for pseudo-labeling are selected among the collocation points (the black dots in Fig. 4a). Locations which are close enough to the data points with labels or pseudo-labels are added to the set which contributes to $\mathcal{L}_{\text{PDE}}$ and $\mathcal{L}_{\text{B}}$ (the black and red dots in Fig. 4b). 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 in a similar way (Fig. 4d-e). 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 $\mathcal{L}_{\text{S}}$;

2. *PINN Ensemble (Ens):* a version without pseudo-labels, in which the points with a high degree of ensemble agreement are used to compute $\mathcal{L}_{\text{PDE}}$ but not $\mathcal{L}_{\text{S}}$.

Each member of the ensemble is trained to minimize the loss in Eq. 4 with shared weights $w_{\text{B}} = 1/|I_{\text{B}}|$, $w_{\text{PDE}} = 1/|I_{\text{PDE}}|$ and $w_{\text{S}} = 1/|D|$ for the PINN Ensemble version and $w_{\text{S}} = 1/|D \cup I_{\text{PDE}}|$ for the version with pseudo-labels. Other weighting strategies (e.g., Wight & Zhao, 2020; Wang et al., 2021a;c) could be combined with our approach as well.

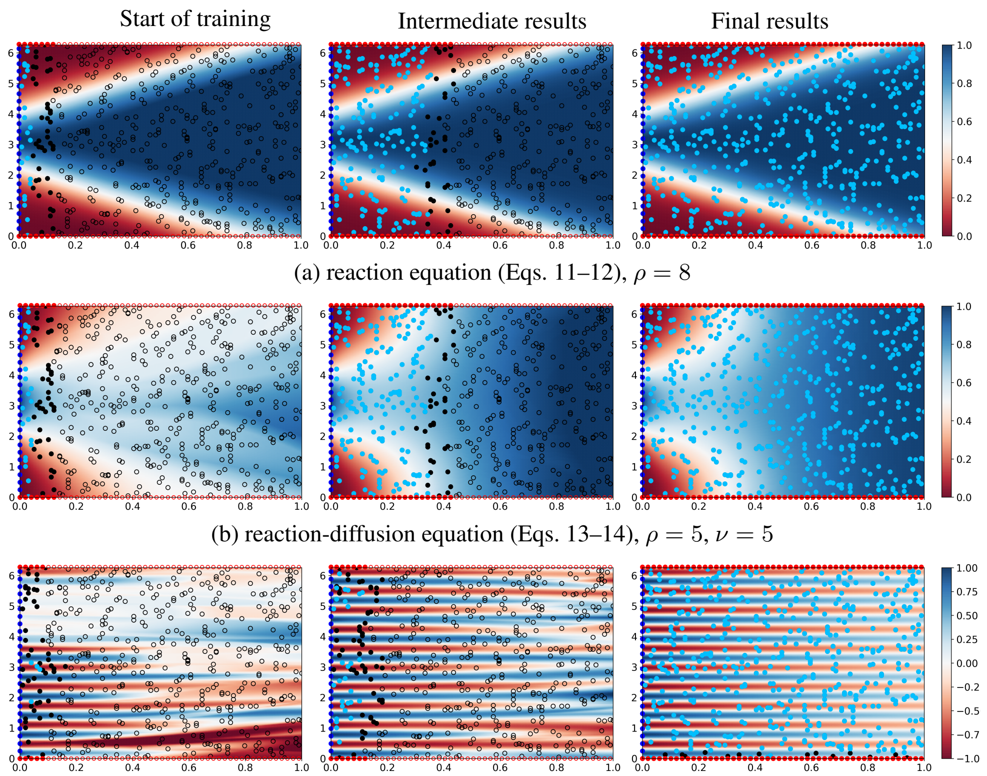
3.3 Related work

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 (Wight & Zhao, 2020; Krishnapriyan et al., 2021; Mattey & Ghosh, 2022) and the adaptive weighting method that respects causality (Wang et al., 2022). 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 (Wight & Zhao, 2020; Krishnapriyan et al., 2021; Mattey & Ghosh, 2022) or an automatic (Wang et al., 2022) 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 (Eqs. 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.

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 (see, e.g, Lee et al., 2013; Sohn et al., 2020): when a classifier becomes confident in the predicted class of an unlabeled example, that example is added to the labeled set. Model ensembles (Laine & Aila, 2017) or prediction ensembles (Berthelot et al., 2019) 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 in Eq. 7. Label propagation in regression tasks is less studied with only a few existing works on semi-supervised regression (Jean et al., 2018; Li et al., 2017). 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.

4 Experiments

We test the proposed algorithm on finding the solutions of the following differential equations:



(a) reaction equation (Eqs. 11–12), $\rho = 8$

(b) reaction-diffusion equation (Eqs. 13–14), $\rho = 5$, $\nu = 5$

(c) diffusion equation with periodic boundary conditions (Eqs. 15–16), $d = 10$

Figure 5: The training progress of PINN ensembles with pseudo-labels. The dot color scheme is from Fig. 4.

- convection equation used to model transport phenomena

$$\frac{\partial u}{\partial t} = -\beta\frac{\partial u}{\partial x}, \quad x \in [0, 2\pi], \quad t \in [0, 1], \quad \beta = \text{const} \tag{9}$$
$$u(x, 0) = \sin(x), \quad u(0, t) = u(2\pi, t) \tag{10}$$

- reaction system for modelling chemical reactions

$$\frac{\partial u}{\partial t} = \rho u(1-u), \quad x \in [0, 2\pi], \quad t \in [0, 1], \quad \rho = \text{const} \tag{11}$$
$$u(x, 0) = \exp\left(-8(x-\pi)^2/\pi^2\right), \quad u(0, t) = u(2\pi, t) \tag{12}$$

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

$$\frac{\partial u}{\partial t} = \nu\frac{\partial^2 u}{\partial x^2} + \rho u(1-u), \quad x \in [0, 2\pi], \quad t \in [0, 1], \quad \nu, \rho = \text{const}, \tag{13}$$
$$u(x, 0) = \exp\left(-8(x-\pi)^2/\pi^2\right), \quad u(0, t) = u(2\pi, t), \quad u_x(0, t) = u_x(2\pi, t) \tag{14}$$

- diffusion equation with periodic boundary conditions

$$\frac{\partial u}{\partial t} = \frac{1}{d^2}\frac{\partial^2 u}{\partial x^2}, \quad x \in [0, 2\pi], \quad t \in [0, 1], \quad d = \text{const} \tag{15}$$
$$u(x, 0) = \sin(dx) \quad u(0, t) = u(2\pi, t), \quad u_x(0, t) = u_x(2\pi, t) \tag{16}$$

- diffusion equation in 15 with boundary conditions of the Dirichlet type:

$$u(0, t) = u(2\pi, t) = 0 \tag{17}$$

Table 1: The means (and standard deviations) across 10 random seeds of the relative l_2 errors in Eq. 18 for the convection (Eqs. 9–10), reaction (Eqs. 11–12) and reaction-diffusion (Eqs. 13–14) systems. All numbers should be multiplied by 10^{-3}. The rows marked with * show the results obtained with $K_{\text{PDE}} = 25600$ collocation points, otherwise $K_{\text{PDE}} = 1000$. In **bold**: the best three results.

Method	Convection, β		Reaction, ρ		Reaction-diffusion, ν	
	$\beta = 30$	$\beta = 40$	$\rho = 5$	$\rho = 7$	$\nu = 3$	$\nu = 4$
PINN, LBFGS	827 (39)	887 (21)	985 (7)	997 (2)	726 (346)	799 (257)
PINN, Adam	13.4 (3.2)	16.2 (4.1)	11.9 (6.2)	887 (269)	252 (375)	437 (433)
PINN, Adam+LBFGS	**9.02 (2.03)**	**13.1 (2.7)**	10.8 (3.5)	967 (26)	170 (328)	426 (427)
SA-PINN	925 (35)	980 (58)	**4.48 (0.94)**	11.5 (2.3)	**5.11 (0.53)**	5.94 (1.63)
Causality	1425 (4)	1393 (3)	41.4 (4.2)	220 (23)	700 (4)	713 (3)
Causality*	436 (496)	1104 (228)	6.01 (1.87)	43.8 (8.0)	6.51 (0.24)	6.56 (0.17)
bc-PINN, Adam	212 (251)	405 (331)	5.37 (2.69)	**8.45 (2.14)**	6.80 (2.70)	8.94 (4.44)
bc-PINN, Adam+LBFGS	21.2 (40.8)	580 (301)	**1.86 (0.81)**	**3.23 (2.99)**	6.65 (0.11)	**6.51 (0.13)**
PL, Adam (our)	21.3 (1.5)	51.0 (4.8)	21.3 (2.5)	55.7 (4.1)	8.35 (0.42)	7.95 (0.47)
PL, Adam+LBFGS (our)	**5.42 (1.23)**	**6.72 (1.37)**	**2.41 (0.44)**	**6.02 (0.85)**	**6.59 (0.01)**	**6.45 (0.03)**
Ens, Adam (our)	**7.00 (2.74)**	**12.6 (3.0)**	7.21 (1.57)	13.4 (3.9)	6.68 (0.37)	6.63 (0.30)
Ens, Adam+LBFGS (our)	8.69 (10.5)	14.0 (11.8)	6.83 (1.34)	7.07 (2.14)	**6.62 (0.01)**	**6.48 (0.02)**

Table 2: The means (and standard deviations) across 10 random seeds of the relative l_2 errors in Eq. 18 for the diffusion system (Eq. 15) with periodic and Dirichlet (Eqs. 16– 17) boundary conditions. All numbers should be multiplied by 10^{-3}. The rows marked with * show the results obtained with $K_{\text{PDE}} = 20000$, otherwise $K_{\text{PDE}} = 1000$. In **bold**: the best three results.

Method	periodic boundary conditions			Dirichlet boundary conditions		
	$d = 5$	$d = 7$	$d = 10$	$d = 5$	$d = 7$	$d = 10$
PINN, LBFGS	997 (.1)	999 (.1)	999 (.1)	998 (.1)	999 (.1)	999 (.1)
PINN, Adam	7.16 (5.61)	20.1 (18.2)	36.6 (18.1)	10.6 (12.0)	15.4 (14.8)	17.9 (10.8)
PINN, Adam+LBFGS	**0.30 (0.04)**	**0.60 (0.42)**	**1.00 (0.27)**	**0.32 (0.07)**	**0.39 (0.08)**	**0.54 (0.08)**
SA-PINN, Adam	5.28 (1.68)	5.94 (1.00)	9.52 (2.61)	4.85 (1.38)	6.76 (1.46)	8.08 (2.69)
SA-PINN, Adam+LBFGS	2.59 (1.07)	3.10 (1.62)	6.46 (3.29)	2.71 (0.96)	3.53 (1.28)	5.68 (2.85)
bc-PINN, Adam	16.4 (6.1)	82.2 (176)	163 (212)	13.6 (6.1)	22.0 (7.6)	50.4 (50.7)
bc-PINN*, Adam	10.7 (5.1)	18.7 (6.9)	48.1 (38.1)	16.9 (12.2)	20.5 (8.8)	29.7 (16.1)
bc-PINN, Adam+LBFGS	4.49 (4.67)	70.0 (190)	108 (257)	3.77 (2.87)	6.14 (2.44)	28.1 (44.2)
PL, Adam (our)	8.26 (3.52)	11.5 (3.4)	27.0 (8.0)	9.15 (3.59)	10.3 (4.0)	25.9 (13.5)
Ens, Adam (our)	5.67 (3.02)	10.1 (4.0)	18.2 (6.7)	8.07 (4.95)	10.1 (5.2)	14.5 (7.3)
PL, Adam+LBFGS (our)	**0.30 (0.05)**	**0.53 (0.04)**	**1.64 (0.59)**	**0.29 (0.06)**	**0.55 (0.07)**	**1.73 (0.58)**
Ens, Adam+LBFGS (our)	**0.12 (0.02)**	**0.41 (0.13)**	**1.59 (0.43)**	**0.11 (0.01)**	**0.33 (0.14)**	**2.03 (1.25)**

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. The models are trained either with the Adam optimizer (Kingma & Ba, 2014) with learning rate 0.001 or with Adam followed by fine-tuning with LBFGS (Liu & Nocedal, 1989). 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.

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 for the considered systems.

To evaluate the accuracy of the proposed algorithm, we compute metrics used in the previous works (Krishnapriyan et al., 2021; Wang et al., 2022): we report the relative l_2 error

$$r = l_2(\hat{\boldsymbol{u}} - \boldsymbol{u})/l_2(\boldsymbol{u}) , \tag{18}$$

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

In Table 1, we compare the accuracy of the proposed method with several strong baselines proposed recently in the literature. We present the means and standard deviations of the solution errors obtained in 10 runs with different initializations. Since most of the considered methods benefit from longer training, we limit the number of network updates for all the comparison methods. We use the following comparison methods: (1) Vanilla PINN trained with the LBFGS optimizer, (2) Vanilla PINN trained with the Adam optimizer, (3) Causality (Wang et al., 2022), (4) SA-PINN (McClenny & Braga-Neto, 2020), (5) bc-PINN (Mattey & Ghosh, 2022). More details on the methods and the selection of hyperparameters can be found in Appendix A.2.

The results in Table 1 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.

In Table 2, we compare the accuracy of the proposed algorithm on solving the diffusion system with the two types of boundary conditions: periodic (Eq. 16) and Dirichlet-type (Eq. 17) boundary conditions. In this comparison, we omit the causality-motivated baseline (Wang et al., 2022) because the periodic boundary conditions in the existing implementations are enforced as hard constraints. The results in Table 2 show that training ensembles of PINNs yields competitive performance in these settings as well.

In Appendix A.3, 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 Δ_{PDE} which determines how far the points considered for inclusion can be from the points already included in the loss calculations. The performance of the algorithm drops when Δ_{PDE} is too large. This happens because ensemble members may be attracted by the same trivial solution in areas too far away from the initial conditions, the effect caused by commonly used network architectures and initialization schemes (see, e.g., Wong et al., 2022; Rohrhofer et al., 2022), which may negatively affect the ensemble diversity.

5 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 Fig. 4c, e). 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 simple systems such as convection and reaction-diffusion.

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 (Hinton et al., 2012) or pseudo-ensembles (Bachman et al., 2014). 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 (Wang et al., 2021c). Another direction is to find alternative ways of the solution interval expansion (e.g., update sets $\mathcal{I}'_{\text{PDE}}$, $\mathcal{I}'_{\text{B}}$ 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 Sanchez-Gonzalez et al., 2020; Iakovlev et al., 2021; Brandstetter et al., 2022), might provide a solution for that.

REFERENCES

Philip Bachman, Ouais Alsharif, and Doina Precup. Learning with pseudo-ensembles. *Advances in Neural Information Processing Systems*, 27, 2014.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. *International Conference on Learning Representations*, 2022.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.

Richard Courant and David Hilbert. *Methods of mathematical physics: partial differential equations*. John Wiley & Sons, Ltd, 1989. ISBN 9783527617234.

Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Rethinking the importance of sampling in physics-informed neural networks. *arXiv preprint arXiv:2207.02338*, 2022.

Suchuan Dong and Naxian Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *Journal of Computational Physics*, 435: 110242, 2021.

Lawrence C Evans. *Partial Differential Equations*, volume 19 of *Graduate studies in mathematics*. American Mathematical Society, 1998. ISBN 9780821807729.

Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. NVIDIA SimNet: An AI-accelerated multi-physics simulation framework. In *International Conference on Computational Science*, pp. 447–461. Springer, 2021.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time PDEs from sparse data with graph neural networks. *International Conference on Learning Representations*, 2021.

Neal Jean, Sang Michael Xie, and Stefano Ermon. Semi-supervised deep kernel learning: Regression with unlabeled data by minimizing predictive variance. *Neural Information Processing Systems*, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

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

Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *International Conference on Learning Representations*, 2017.

Dong-Hyun 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*, volume 3, pp. 896, 2013.

Yu-Feng Li, Han-Wen Zha, and Zhi-Hua Zhou. Learning safe prediction for semi-supervised regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.

Dehao Liu and Yan Wang. A dual-dimer method for training physics-constrained neural networks with minimax architecture. *Neural Networks*, 136:112–125, 2021.

Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021a.

Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021b.

Stefano Markidis. The old and the new: Can physics-informed deep-learning replace traditional linear solvers? *Frontiers in big Data*, pp. 92, 2021.

Revanth Mattey and Susanta 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*, 390:114474, 2022. ISSN 0045-7825.

Levi McClenny and Ulisses Braga-Neto. Self-adaptive physics-informed neural networks using a soft attention mechanism. *arXiv preprint arXiv:2009.04544*, 2020.

Maziar Raissi, Paris Perdikaris, and George Em 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.

Franz M Rohrhofer, Stefan Posch, Clemens Gößnitzer, and Bernhard C Geiger. Understanding the difficulty of training physics-informed neural networks on dynamical systems. *arXiv preprint arXiv:2203.13648*, 2022.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7462–7473. Curran Associates, Inc., 2020.

Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in Neural Information Processing Systems*, 33:596–608, 2020.

N Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7537–7547. Curran Associates, Inc., 2020.

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

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

Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, pp. 110768, 2021c.

Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404*, 2022.

Colby L Wight and Jia Zhao. Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks. *arXiv preprint arXiv:2007.04542*, 2020.

Jian Cheng Wong, Chinchun Ooi, Abhishek Gupta, and Yew-Soon Ong. Learning in sinusoidal spaces with physics-informed neural networks. *IEEE Transactions on Artificial Intelligence*, 2022.

Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *arXiv preprint arXiv:2207.10289*, 2022.

Kirill Zubov, Zoe McCarthy, Yingbo Ma, Francesco Calisto, Valerio Pagliarino, Simone Azeglio, Luca Bottero, Emmanuel Luján, Valentin Sulzer, Ashutosh Bharambe, et al. NeuralPDE: Automating physics-informed neural networks (PINNs) with error approximations. *arXiv preprint arXiv:2107.09443*, 2021.

A Appendix

A.1 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., Wang et al., 2021a; Krishnapriyan et al., 2021; Wang et al., 2021c;b; 2022; Wight & Zhao, 2020). 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 (see, e.g., Markidis, 2021): the weights w_S, w_B, w_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_PDE of the individual loss terms in Eqs. 5–7 is a popular way to improve the accuracy of the PINN solution. Wight & Zhao (2020) propose using larger weights w_S relative to w_B and w_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_PDE during training such that the weights that correspond to problematic loss terms get higher values. Wang et al. (2021a;c) propose to adjust the weights either based on the magnitudes of the gradients of the corresponding loss terms or based on the eigenvalues of the limiting Neural Tangent Kernel. Liu & Wang (2021) propose to treat weights w_S, w_B, w_PDE as trainable parameters and adjust them jointly with the PINN parameters solving a minimax optimization problem. Self-Adaptive PINNs (SA-PINNs, McClenny & Braga-Neto, 2020) increase *point-wise* weights w_i, w_j and w_k of the loss terms in Eqs. 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. Several works (see, e.g., Lagaris et al., 1998; Dong & Ni, 2021; Lu et al., 2021b; Sukumar & Srivastava, 2022) propose neural network parameterizations which guarantee that the initial or boundary conditions are satisfied exactly, thus eliminating terms $\mathcal{L}_\text{S}$ and $\mathcal{L}_\text{B}$ from Eq. 4. Wang et al. (2021b) propose to use Fourier features as the inputs of the neural network, which is motivated by the success of Fourier feature networks (Tancik et al., 2020) in preserving high-frequencies in the modeled solution. The method called SIREN (Sitzmann et al., 2020) 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. Wang et al. (2021a) propose to add multiplicative connections to the standard multilayer perceptron architecture 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 $\mathcal{L}_{\text{PDE}}$ in Eq. 7. Wight & Zhao (2020) propose to sample more collocation points in the areas with large values of $\mathcal{L}_{\text{PDE}}$, which helps to learn solutions with fast transitions. Daw et al. (2022) propose an evolutionary strategy for sampling the collocation points: the points with large contribution to $\mathcal{L}_{\text{PDE}}$ 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 (Wu et al., 2022).

Many of the improvements proposed to PINNs can be combined, which is supported by existing software libraries (Lu et al., 2021a; Zubov et al., 2021; Hennigh et al., 2021).

A.2 Training details

For all comparison methods (similarly to Krishnapriyan et al. (2021)), we use $K_{\text{PDE}} = 1000$ collocation points randomly sampled from a regular 256×100 grid on the solution interval. We use $K_{\text{S}} = 256$ points to fit the initial conditions, these points are selected from a regular grid on $x \in [0, 2\pi]$ with $t = 0$. We use $K_{\text{B}} = 100$ for the boundary condition loss $\mathcal{L}_{\text{BC}}$. These points are selected form a regular grid on $t \in [0, 1]$.

For the proposed ensemble method, we use the following hyperparameters: $\sigma^2 = 0.0004$, $\epsilon = 0.001$, $\Delta = 0.05$ and $\Delta_{\text{PDE}} = 0.1$. We perform $N = 1000$ gradient updates before extending sets I'_{PL}, I'_{PDE}, I_{B}, except for the first set extension which is done after $N_1 = 5000$ training steps. We also set weighting coefficient $w_s = 64/|D \cup I_{\text{PDE}}|$ for runs with pseudo-labels trained with Adam and LBFGS similar to the closest baseline.

For the baseline methods we adapted the authors' implementations with the following adjustments.

- *Causality (Wang et al., 2022):* PINN with an adaptive weighting scheme that respects causality. We report results for the values of hyperparameter ϵ in Eq. 8 that worked best for the considered systems: $\epsilon = 0.01$ for the convection system, $\epsilon = 100$ for the reaction system and $\epsilon = 1$ for the reaction-diffusion systems. We train the model until all the adapted weights become greater than 0.99 (the stopping criterion used by Wang et al. (2022)). As the existing implementation requires a regular grid, we use a grid of $K_{\text{PDE}} = 32 \times 32 = 1024$ collocation points to compute loss $\mathcal{L}_{\text{PDE}}$. We also report results with a denser grid of $K_{\text{PDE}} = 256 \times 100 = 25600$ points for this algorithm. We do not add normalization for network inputs x and t to $[-1, 1]$ and follow input encoding of the original implementation.

- *SA-PINN (McClenny & Braga-Neto, 2020)* 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.

- *bc-PINN (Mattey & Ghosh, 2022)*: a technique of expanding the time interval with a predefined schedule. We expand the time interval four times and sample 250 collocation points on each interval to compute loss $\mathcal{L}_{\text{PDE}}$ and 25 points per interval to compute $\mathcal{L}_{\text{B}}$. We report the results obtained with the Adam optimizer and with the Adam optimizer followed by LBFGS on each time interval.

For all of the methods, we train convection equation (Eqs. 9–10) with $\beta = 30$ for 104000 gradient updates and with $\beta = 40$ for 154000 updates, reaction (Eqs. 11–12) and reaction-diffusion (Eqs. 13–14) equations are trained for 64000 updates, diffusion (Eqs. 15–17) equation with $d = 5$ and $d = 7$ for 84000 updates and with $d = 10$ for 104000 (unless an automatic stopping criterion is used).

A.3 Hyperparameter sensitivity

In Table 3, we test the sensitivity of the ensemble training of PINNs to its hyperparameters. In these experiments, we use the number of updates as described in Appendix A.2. We consider that a run has not converged if less than 95% of the sampled points have been added to the set which is used to compute $\mathcal{L}_{\text{PDE}}$ by the last update. Such runs are excluded from the statistics reported in Table 3. This step is done only in the hyperparameter sensitivity experiments.

The most important hyperparameters are the variance threshold σ^2 and the distance parameters Δ and Δ_{PDE} which determine how quickly the algorithm expands the solution interval. When σ^2 is small then the interval is expanded more slowly and the algorithm may require more iterations to converge. Too large values of σ^2 can result in adding new regions in the training procedure too early. The distance hyperparameters Δ and Δ_{PDE} have a similar effect. However, for the considered systems, the model performance is stable with little sensitivity to the values of these hyperparameters as well as to the number K_{PDE} of collocation points and the number of networks in the ensemble.

Table 3: The results obtained with the proposed 'PINN Ensemble' algorithm using different hyperparameter values. We report the same metric for the same systems as in Table 1. The superscript $^{(n)}$ indicates how many runs among the 10 runs did not converge after a fixed number of updates. We assume that the algorithm has converged if more than 95% of the sampled points have been added to the set used to compute loss $\mathcal{L}_{\text{PDE}}$.

	Convection, β		Reaction, ρ			Reaction-diffusion, ν		
	$\beta = 30$	$\beta = 40$	$\rho = 5$	$\rho = 6$	$\rho = 7$	$\nu = 2$	$\nu = 3$	$\nu = 4$
default	7.00 (2.7)	12.6 (3)	7.21 (1.6)	11.0 (2)	13.4 (4)	6.93 (.3)	6.68 (.4)	6.63 (.3)
Varying the threshold for the ensemble disagreement, default $\sigma^2 = 0.0004$								
$\sigma^2 = 10^{-4}$	8.15 (2.7)$^{(1)}$	14.0 (2)	7.16 (1.6)	11.3 (2)	12.3 (3)$^{(3)}$	6.67 (.4)	6.73 (.6)	6.60 (.5)
$\sigma^2 = 10^{-3}$	8.11 (2.0)	14.5 (4)	7.16 (1.5)	11.3 (2)	13.0 (4)	6.68 (.3)	6.79 (.4)	6.63 (.4)
Varying the distances which determinate candidate collocation points, default $\Delta_{\text{PDE}} = 0.1$, $\Delta = 0.05$								
$\Delta_{\text{PDE}} = .07$	10.1 (4.4)	17.6 (10)$^{(2)}$	6.80 (1.1)	11.5 (3)	10.7 (3)$^{(2)}$	7.03 (.3)	6.76 (.6)	6.68 (.6)
$\Delta_{\text{PDE}} = .2$	5.33 (0.9)	11.5 (2)	7.61 (2.)	12.7 (3)	11.6 (4)	7.13 (.4)	6.89 (.4)	6.49 (.4)
$\Delta = .07$	7.82 (1.6)	13.2 (2)	7.67 (1.5)	12.0 (2)	15.3 (4)	6.70 (.6)	6.64 (.5)	6.63 (.4)
$\Delta_{\text{PDE}} = .25$ & $\Delta = .125$	6.25 (1.6)	13.0 (2)	8.42 (2.)	14.2 (3)	14.5 (8)	6.97 (.4)	6.88 (.6)	6.75 (.4)
$\Delta = 10^5$	8.98 (1.9)	13.7 (2)	8.33 (1.6)	13.6 (2)	18.4 (3)	6.69 (.2)	6.91 (.6)	6.62 (.4)
$\Delta_{\text{PDE}} = 10^5$	*11.1 (1.2)*	*16.9 (3)*	*8.22 (1.5)*	*56 (71)*	*36.3 (53)*	*455 (143)*	*242 (164)*	*74 (82)*
Varying the prediction error which determinate candidate collocation points, default $\epsilon = 0.001$								
$\epsilon = 5 \cdot 10^{-4}$	8.94 (2.3)	13.9 (6)	7.13 (1.5)	11.2 (2)	12.9 (4)$^{(1)}$	6.66 (.3)	6.93 (.7)	6.61 (.4)
$\epsilon = 10^{-2}$	7.65 (1.7)	12.3 (2)	7.15 (1.5)	11.4 (2)	13.0 (4)$^{(1)}$	6.72 (.4)	6.93 (.7)	6.63 (.3)
$\epsilon = 10^5$	7.19 (1.4)	12.2 (2)	7.11 (1.5)	11.4 (2)	12.9 (4)$^{(1)}$	6.72 (.4)	6.93 (.7)	6.61 (.4)
Varying the number of the collocation points, default $K_{\text{PDE}} = 1000$								
$K_{\text{PDE}} = 5000$	8.02 (1.9)	14.1 (2)	6.75 (1.4)	10.7 (2)	10.4 (3)	6.88 (.4)	6.52 (.5)	6.55 (.5)
$K_{\text{PDE}} = 10^4$	7.71 (1.7)	16.8 (4)	7.00 (1.3)	9.89 (2)	10.3 (4)	6.94 (.4)	6.71 (.5)	6.55 (.4)
Number of training epochs for the first iteration, default $N_1 = 5000$								
$N_1 = 10^3$	7.12 (1.0)	15.6 (3)	7.86 (1.3)	12.6 (2)	14.8 (2)$^{(2)}$	6.89 (.3)	6.45 (.4)	6.58 (.2)
$N_1 = 10^4$	10.8 (7)	15.0 (8)$^{(1)}$	6.26 (1.4)	9.18 (2)	9.73 (3.2)	6.79 (.3)	6.67 (.4)	6.66 (.5)
Number of training epochs between adding new points, default $N = 1000$								
$N = 700$	7.64 (1.4)	15.1 (7)	7.53 (1.7)	12.1 (2)	14.3 (3)$^{(1)}$	6.91 (.4)	6.66 (.4)	6.65 (.2)
$N = 2000$	9.95 (1.7)	14.4 (5)	6.62 (1.3)	9.99 (2)	11.8 (3)	6.96 (.4)	6.87 (.4)	6.54 (.5)
Number of networks, default $L = 5$								
$L = 3$	8.96 (3.7)	12.2 (4)	6.45 (1.4)	11.6 (4)	14.4 (6)$^{(3)}$	6.83 (.3)	6.60 (.3)	6.75 (.6)
$L = 7$	6.68 (1.6)	12.3 (2)	6.07 (.81)	11.4 (2)	11.9 (3)$^{(5)}$	6.88 (.2)	6.78 (.4)	6.57 (.2)
$L = 10$	7.94 (1.4)	13.6 (2)	6.48 (.58)	10.6 (.9)	12.5 (3)	7.05 (.3)	6.72 (.1)	6.59 (.2)

A.4 More experimental results

In this section we report additional experimental results for reaction (Eqs. 11–12) with $\rho = 6$ and reaction-diffusion (Eqs. 13–14) with $\nu = 2$.

Table 4: The means (and standard deviations) across 10 random seeds of the relative l_2 errors in Eq. 18 for reaction (Eqs. 11–12) and reaction-diffusion (Eqs. 13–14) systems. All numbers should be multiplied by 10^{-3}. The rows marked with * show the results obtained with $K_{\text{PDE}} = 25600$ collocation points otherwise $K_{\text{PDE}} = 1000$. In **bold**: the best three results.

Method	Reaction, ρ	Reaction-diffusion, ν
	$\rho = 6$	$\nu = 2$
PINN, LBFGS	993 (2)	608 (469)
PINN, Adam	592 (459)	7.01 (0.74)
PINN, Adam+LBFGS	678 (433)	**6.83 (0.03)**
SA-PINN	8.12 (1.40)	6.97 (0.54)
Causality	102 (35)	678 (5)
Causality*	18.9 (4.1)	7.06 (0.27)
bc-PINN, Adam	**5.66 (2.02)**	7.11 (1.35)
bc-PINN, Adam+LBFGS	**2.96 (1.25)**	6.89 (0.08)
PL, Adam (our)	38.5 (3.5)	8.83 (0.42)
PL, Adam+LBFGS (our)	**4.17 (0.58)**	**6.83 (0.02)**
Ens, Adam (our)	11.0 (2.3)	6.93 (0.32)
Ens, Adam+LBFGS (our)	9.73 (2.02)	**6.85 (0.02)**