# iPINNs: incremental learning for Physics-informed neural networks

Dekhovich, Aleksandr; Sluiter, Marcel H.F.; Tax, David M.J.; Bessa, Miguel A.

**DOI**
10.1007/s00366-024-02010-1

**Publication date**
2024

**Document Version**
Final published version

**Published in**
Engineering with Computers

Engineering with Computers
https://doi.org/10.1007/s00366-024-02010-1

ORIGINAL ARTICLE

# iPINNs: incremental learning for Physics-informed neural networks

**Aleksandr Dekhovich**[1] · **Marcel H. F. Sluiter**[1] · **David M. J. Tax**[2] · **Miguel A. Bessa**[3]

Received: 23 November 2023 / Accepted: 5 June 2024


**Abstract**
Physics-informed neural networks (PINNs) have recently become a powerful tool for solving partial differential equations (PDEs). However, finding a set of neural network parameters that fulfill a PDE at the boundary and within the domain of interest can be challenging and non-unique due to the complexity of the loss landscape that needs to be traversed. Although a variety of multi-task learning and transfer learning approaches have been proposed to overcome these issues, no incremental training procedure has been proposed for PINNs. As demonstrated herein, by developing incremental PINNs (iPINNs) we can effectively mitigate such training challenges and learn multiple tasks (equations) sequentially without additional parameters for new tasks. Interestingly, we show that this also improves performance for every equation in the sequence. Our approach learns multiple PDEs starting from the simplest one by creating its own subnetwork for each PDE and allowing each subnetwork to overlap with previously learned subnetworks. We demonstrate that previous subnetworks are a good initialization for a new equation if PDEs share similarities. We also show that iPINNs achieve lower prediction error than regular PINNs for two different scenarios: (1) learning a family of equations (e.g., 1-D convection PDE); and (2) learning PDEs resulting from a combination of processes (e.g., 1-D reaction–diffusion PDE). The ability to learn all problems with a single network together with learning more complex PDEs with better generalization than regular PINNs will open new avenues in this field.

**Keywords** Physic-informed neural networks (PINNs) · Scientific machine learning (SciML) · Incremental learning · Sparsity

## 1 Introduction

Deep neural networks (DNNs) play a central role in scientific machine learning (SciML). Recent advances in neural networks find applications in real-life problems in physics [1–4], medicine [5–7], finance [8–11], and engineering [12–15]. In particular, they are also applied to solve Ordinary Differential Equations and Partial Differential Equations (ODEs/PDEs) [16–20]. Consider the following PDE,

$$\mathcal{F}[u(\mathbf{x},t)] = f(\mathbf{x}), \quad \mathbf{x} \in \Omega,\ t \in (t_0, T], \tag{1}$$

$$\mathcal{B}[u(\mathbf{x},t)] = b(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega, \tag{2}$$

$$u(\mathbf{x},t_0) = h(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{3}$$

where $\mathcal{F}$ is a differential operator, $\mathcal{B}$ is a boundary condition operator, $h(\mathbf{x})$ is an initial condition, and $\Omega$ is a bounded domain.

The first neural network-based approaches incorporated a form of the equation into the loss function with initial and boundary conditions included as hard constraints [21, 22]. However, these works used relatively small neural networks with one or two hidden layers. On the contrary, PINNs [20] encode initial and boundary conditions as soft constraints into the loss function of a DNN. Subsequently, PINNs and their extensions found applications in fluid mechanics [23–25], inverse problems [26–28] and finance [20, 29]. Later, the generalized version of PINNs, called XPINNs [30], was proposed by decomposing the domain into multiple subdomains. However, this method uses as

✉ Miguel A. Bessa
miguel_bessa@brown.edu

[1] Department of Materials Science and Engineering, Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands

[2] Pattern Recognition and Bioinformatics Laboratory, Delft University of Technology, Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

[3] School of Engineering, Brown University, 184 Hope St., Providence, RI 02912, USA

Published online: 22 June 2024

many networks as the number of subdomains, increasing the algorithm's complexity. Almost simultaneously with our work, Multi-head PINNs (MH-PINNs) [31] have been proposed as a multi-task and meta-learning approach for PINNs that is employed to learn stochastic processes, synergistic learning of PDEs and uncertainty quantification. MH-PINNs have a shared part of the network and task-specific output heads for prediction. Therefore, it uses additional parameters for every head, increasing the model's size with respect to the number of tasks, without sharing knowledge between them. In addition, the parameters in MH-PINN are shared between all tasks in the non-output layer, which is a limitation if the tasks are very different as the authors noted [31]. Other meta-learning approaches were also employed in the context of PINNs [32, 33]. However, meta-learning literature focuses on obtaining good initialization for a new task given some tasks for pretraining. Unfortunately, once the network is adapted to a new task, it loses the ability to solve the previous ones, i.e. it undergoes *catastrophic forgetting* of other tasks. We take a different route inspired by the incremental learning and continual (or lifelong) learning literature, as discussed below.

**Background and main challenges** PINNs formulate the PDE solution problem by including initial and boundary conditions into the loss function of a neural network as soft constraints. Let us denote the output of the network $\mathcal{N}$ with learnable parameters $\theta$ as $\hat{u}(\theta, \mathbf{x}, t) = \mathcal{N}(\theta; \mathbf{x}, t)$. Then sampling the set of collocation points, i.e. a set of points in the domain, $\mathcal{CP} = \{(x^i, t^i) : x^i \in \text{int}\,\Omega,\ t^i \in (t_0, T],\ i = 1, 2, \dots N_{\mathcal{F}}\}$, the set of initial points $\mathcal{IP} = \{(x^j, t_0) : x^j \in \partial\Omega,\ j = 1, 2, \dots, N_{u_0}\}$ and the set of boundary points $\mathcal{BP} = \{(x^k, t^k) : x^k \in \partial\Omega,\ t^k \in (t_0, T],\ k = 1, 2, \dots, N_b\}$ one can write the optimization problem and loss function arising from PINNs as follows:

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathcal{F}}(\theta) + \mathcal{L}_{u_0}(\theta) + \mathcal{L}_b(\theta) \to \min_{\theta}, \tag{4}$$

$$\mathcal{L}_{\mathcal{F}}(\theta) = \frac{1}{N_{\mathcal{F}}} \sum_{i=1}^{N_{\mathcal{F}}} \left|\left|\mathcal{F}[\hat{u}(\theta, x^i, t^i)] - f(x^i)\right|\right|^2, \quad (x^i, t^i) \in \mathcal{CP}, \tag{5}$$

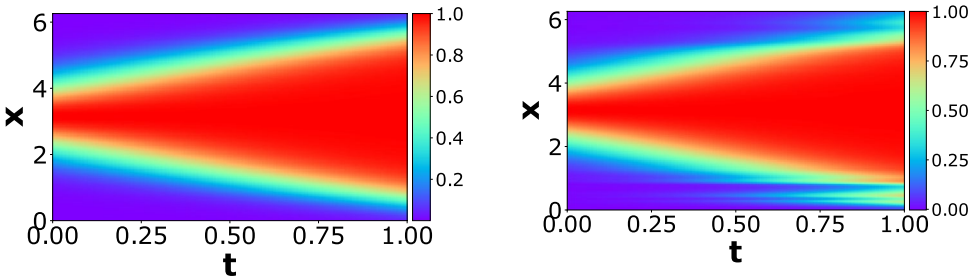
$$\mathcal{L}_{u_0}(\theta) = \frac{1}{N_{u_0}} \sum_{j=1}^{N_{u_0}} \left|\left|\hat{u}(\theta, x^j, t_0) - h(x^j)\right|\right|^2, \quad (x^j, t_0) \in \mathcal{IP}, \tag{6}$$

$$\mathcal{L}_b(\theta) = \frac{1}{N_b} \sum_{k=1}^{N_b} \left|\left|\mathcal{B}[\hat{u}(\theta, x^k, t^k)] - b(x^k)\right|\right|^2, \quad (x^k, t^k) \in \mathcal{BP}. \tag{7}$$

However, sometimes PINNs struggle to learn the ODE/PDE dynamics [34–37] (see Fig. 1). Wight and Zhao [38] proposed several techniques to improve the optimization process compared to the original formulation: mini-batch optimization and adaptive sampling of collocation points. Adaptive sampling in time, splits the time interval $[t_0, T] = \cup_{k=0}^{K} [t_{k-1}, t_k]$, $t_K = T$, and solves an equation on the first interval $[t_0, t_1]$, then on $[t_0, t_2]$, and so on up to $[t_0, T]$. Thus, if a solution can be found on a domain $\Omega \times [t_0, t_{k-1}]$, then the network is pretrained well for the extended domain $\Omega \times [t_0, t_k]$. Krishnapriyan et al. [35] proposed the *seq2seq* approach that splits the domain into smaller subdomains in time and learns the solution on each of the subdomains with a separate network. Thus, both adaptive sampling in time and *seq2seq* are based on the idea of splitting the domain into multiple subdomains, on which solutions can be learned more easily.

As explained in [36], improving PINN's solutions by considering small subdomains is possible because the loss residuals ($\mathcal{L}_{\mathcal{F}}$ term) can be trivially minimized in the vicinity of fixed points, despite corresponding to nonphysical system dynamics that do not satisfy the initial conditions. Therefore, the reduction of the domain improves the convergence of the optimization problem (4) and helps to escape nonphysical solutions.

Despite the popularity of DNNs, and PINNs in particular, there are few *incremental* learning algorithms available in SciML literature. Yet, incremental learning and continual




(a) Exact solution. (b) PINNs solution.

**Fig. 1** 1-D reaction equation with parameter $\rho = 5$ (see P1.2)

learning algorithms [39–41] are capable of handling tasks sequentially, instead of altogether as in multi-task learning and other strategies. Moreover, they are still capable of not forgetting how to solve all of the previously learned tasks. If tasks have some similarities with each other, new tasks have the potential of being learned better (i.e., faster or with lower testing error) with the help of previously learned ones. The goal of this work is to propose an incremental learning algorithm for PINNs such that similar symbiotic effects can be obtained.

**Our contribution** We propose *incremental PINNs* (iPINNs) and implement this strategy by creating one subnetwork per task such that a complete neural network can learn multiple tasks sequentially without forgetting of previous tasks. Each subnetwork $\mathcal{N}_i$ has its own set of parameters $\theta_i \subset \theta$, and the model is trained sequentially on different tasks. A subnetwork for a new task can overlap with all previous subnetworks, which helps to assimilate the new task. As a result, the network consists of overlapping subnetworks, while the free parameters can be used for future tasks. To illustrate the benefits of the algorithm we consider two problem formulations (Sect. 3). Firstly, we learn a family of equations (e.g., convection) starting from a simple one and incrementally learning new equations from that family. Secondly, we learn a dynamical system that consists of two processes (e.g., reaction–diffusion) by first learning the individual components of the process. Both scenarios demonstrate that the incremental approach enables an iPINN network to learn for cases where regular PINNs fail. To the best of our knowledge, this is the first example where one network can sequentially learn multiple equations without extending its architecture and be able to provide solutions to all previously seen PDEs without forgetting them, with the added benefit that performance is significantly improved.

## 2 Related work

Our methodology is based on creating sparse network representations and, similarly to other PINN research, is sensitive to the choice of activation functions. We briefly highlight key related work herein.

**Sparse network representation** Sparse architectures are often advantageous compared to dense ones [42–45]. According to the lottery ticket hypothesis (LTH) [46], every randomly initialized network contains a subnetwork that can be trained in isolation to achieve comparable performance as the original network. Based on this observation, the idea of using subnetworks has been adopted in continual learning [47–49]. In this paradigm, every subnetwork created is associated with a particular task and used only for this task to make a prediction. One of the approaches to find these tasks-related subnetworks is connections' pruning [50–54] that removes unimportant parameters while exhibiting similar performance.

**Choice of the activation function** There are several studies that investigate how different activation functions affect the performance of neural networks in classification and regression tasks [55, 56]. It was shown that `ReLU` [57] activation function which can be powerful in classification tasks, in the case of physics-informed machine learning (PIML) regression, may not be the optimal choice. Meanwhile, hyperbolic tangent (`tanh`) or sine (`sin`) perform well for PIML. Sinusoidal representation networks (SIRENs) [58] tackle the problem of modeling the signal with fine details. Special weights initialization scheme combined with `sin` activation function allows SIREN to learn complex natural signals. Hence, we use `sin` activation function in our experiments. In Sect. 6.1, we provide the comparison in results between the discussed activation functions.

**Curriculum and transfer learning** One possible approach for mitigating training difficulties in PINNs is transfer learning which is commonly used in computer vision and natural language processing [59–62]. It tries to improve the optimization process by starting with better weight initialization. In PINNs, transfer learning is also successfully used to accelerate the loss convergence [63–66]. For instance, Chen et al. [67] apply transfer learning to learn different PDEs faster by creating tasks and changing coefficients or source terms in equations. Analogously, curriculum regularization (similar to curriculum learning [68]) is proposed in [35] to find good initial weights. However, in all these scenarios the PINN experiences forgetting, i.e. it loses the ability to generalize on the tasks used in pretraining. The proposed iPINN does not have this issue, maintaining the ability to remember solutions for all given PDEs.

## 3 Problem formulation

We focus on two scenarios: (1) incremental PINNs learning, where the network sequentially learns several equations from the same family; and (2) learning a combination of multiple equations that create another physical process. To illustrate these cases, we consider one-dimensional convection, reaction and reaction–diffusion problems with periodic boundary conditions.

### 3.1 Scenario 1: Equation incremental learning

We consider the problem of learning the sequence of equations that belong to one family:

$$\mathcal{F}_k[u(x,t)] = 0, \quad x \in \Omega,\ t \in [t_0, T],\ k = 1, 2, \ldots, \tag{P1}$$

where $\mathcal{F}_k$, $k = 1, 2, \ldots$ are differential operators from the same family of equations.

**1-D convection equation**

$$\begin{aligned} &\frac{\partial u}{\partial t} + \beta_k \frac{\partial u}{\partial x} = 0, \qquad \text{(P1.1)} \\ &u(x,0) = h_1(x), \\ &u(0,t) = u(2\pi,t), \end{aligned}$$

**1-D reaction equation**

$$\begin{aligned} &\frac{\partial u}{\partial t} - \rho_k u(1-u) = 0, \qquad \text{(P1.2)} \\ &u(x,0) = h_2(x), \\ &u(0,t) = u(2\pi,t), \end{aligned}$$

In this case, every task $k$ is associated with $\mathcal{D}_k = \{(x,t,k) : x \in [0,2\pi],\ t \in [t_0,T],\ k \in \mathbb{N}\}$. Following [35], we take $h_1(x) = \sin x$ and $h_2(x) = e^{-\frac{(x-\pi)^2}{2(\pi/4)^2}}$.

### 3.2 Scenario 2: Combination of multiple equations

We also consider the case when a dynamic process consists of multiple components. Let us consider the reaction–diffusion equation:

$$\begin{aligned} &\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} - \rho u(1-u) = 0, \\ &u(x,0) = h_2(x), \\ &u(0,t) = u(2\pi,t), \end{aligned} \tag{P2}$$

where $t \in [0,1]$, $x \in [0,2\pi]$, $\nu, \rho > 0$. This process consists of two parts: reaction term ($\nu = 0$): $-\rho u(1-u)$ and diffusion term ($\rho = 0$): $-\nu \frac{\partial^2 u}{\partial x^2}$. Therefore, we construct one task as the reaction, another one as the diffusion, and the final one as the reaction–diffusion. We can change the order of the reaction tasks and diffusion tasks to show the robustness of incremental learning. The reaction–diffusion task should be the last one since our goal is first to learn the components of the system and only then the full system.

Considering these two problems, we want to show that better generalization can be achieved by pretraining the network with simpler related problems rather than by dividing the domain into smaller subdomains. In the following section, we show how one network can incrementally learn different equations without catastrophic forgetting.

## 4 Proposed method

The proposed method needs to be applicable to both types of problems P1 and P2. However, these problems cannot be solved by one network with the same output head for every different task, since $\mathcal{F}_i[u(x,t)] \neq \mathcal{F}_j[u(x,t)]$ for $i \neq j$ and $x \in \Omega$, $t \in (t_0, T]$. Instead, the incremental learning algorithm we propose (iPINNs) focuses on learning task-specific subnetworks $\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_k, ...$ for each task $k$. By creating PDE-specific subnetworks we can encode multiple solution in one network since not all parameters are shared between PDEs, and as a result we parameterize a solution of equation $k$ with its own subnetwork $\mathcal{N}_k$. Moreover, we achieve the ability to learn multiple PDEs without extension of the underlying architecture or using multiple networks.

We start by creating the above-mentioned subnetworks using an iterative pruning algorithm that we developed called NNrelief [54]. Other pruning strategies could be considered, without loss of generality—see Remark 1.

***Remark 1*** In principle, any connections pruning algorithm or any other approach that is able to find and train sparse network representations is suitable for the iPINNs strategy we propose herein. However, most pruning algorithms aim at reducing memory requirements or reducing inference time, instead of aiming at subnetwork creation with the smallest number of neuron connections. NNrelief was developed with this in mind, so it creates sparser subnetworks for a given performance level when compared to state-of-the-art methods, as we showed in the original article [54] for multiple datasets. This makes it particularly interesting for iPINNs, as the subnetworks we generate are smaller and leave additional free connections for subsequent incremental training.

NNrelief consists of three steps: (1) *training* the network on the current task; (2) *pruning* unimportant connections based on a proposed criterion (importance scores, as discussed next and also detailed in the original article); and (3) *retraining* the network to obtain satisfactory performance. Steps 2 and 3 (pruning and retraining) can be repeated more than once, although in our experience it is not necessary to repeat it more than 3 times for the network to achieve similar or better performance than before pruning it [54]. NNrelief achieves the highest number of pruned parameters (connections) reported to date for different state-of-the-art

neural network architectures trained on MNIST [50], CIFAR-10/100 [69] and Tiny-ImageNet [70]. The method has the particular characteristic of using input data to estimate the contribution of every connection to the neuron in the pretrained network and then deleting the least important connections.

For an output signal $\mathcal{Y}^{(l-1)} = \{\mathbf{y}_1^{(l-1)}, \mathbf{y}_2^{(l-1)}, \ldots, \mathbf{y}_N^{(l-1)}\}$ corresponding to $N$ samples, where $\mathbf{y}_n^{(l-1)} = (y_{n1}^{(l-1)}, y_{n2}^{(l-1)}, \ldots, y_{nm}^{(l-1)})^t \in \mathbb{R}^m$ and $m$ is the number of neurons in layer $l-1$, NNrelief computes the average strength of a signal that passes through every connection. Thus, the average signal strength between neuron $i$ of layer $l-1$ and neuron $j$ of layer $l$ is computed as follows: $\overline{\left|w_{ij}^{(l)} y_i^{(l-1)}\right|} := \frac{1}{N}\sum_{n=1}^{N}\left|w_{ij}^{(l)} y_{ni}^{(l-1)}\right|$, where $w_{ij}^{(l)}$ is a weight parameter for the corresponding connection. Then the importance score for this connection is defined as:

$$s_{ij}^{(l)} = \frac{\overline{\left|w_{ij}^{(l)} y_i^{(l-1)}\right|}}{\sum_{k=1}^{m}\overline{\left|w_{kj}^{(l)} y_k^{(l-1)}\right|} + \left|b_j^{(l)}\right|}, \tag{8}$$

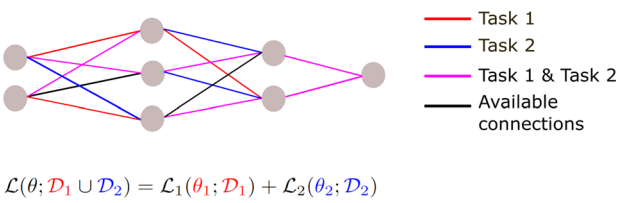



**Fig. 2** An example of iPINNs with two PDEs: every subnetwork corresponds to only one task (PDE). Colors represent belonging of connections to different tasks (PDEs): red for task 1, blue for task 2, magenta for both tasks, and black for not assigned connections (colour figure online)

where $b_j^{(l)}$ is a bias parameter for neuron $j$ of layer $l$. For every neuron $j$, connections $i^*$ with the smallest value of the importance score $s_{i^*j}^{(l)}$ are deleted. The fraction of pruned parameters is defined with $\alpha \in (0, 1)$, where the sum of the remaining importance $s_{ij}^{(l)}$ is at least $\alpha$. Therefore, the smaller value of $\alpha$ results in a higher number of pruned parameters. The pseudocode is shown in Algorithm 1.

**Algorithm 1** Pseudocode for NNrelief

**Require:** network $\mathcal{N}$, training dataset $\mathcal{D} = \{\mathbf{y}_i = (x_i, t_i), i = 1, 2, \ldots, N\}$, pruning hyperparameter $\alpha$.

$\mathcal{Y}^{(0)} \leftarrow \mathcal{D}$
**for** every layer $l = 1, \ldots, L$ **do**
  $\mathcal{Y}^{(l)} \leftarrow \text{layer}(\mathcal{Y}^{(l-1)})$
  **for** every neuron $j$ in layer $l$ **do**
    Compute importance scores $s_{ij}^{(l)}$ for every incoming connection $w_{ij}$ and bias $b_j$ using Eq. 8.
    $\hat{s}_{ij}^{(l)} \leftarrow \text{Sort}(s_{ij}^{(l)}, order = descending)$.
    Find $p_0 = \min\{p : \sum_{i=1}^{p} \hat{s}_{ij}^{(l)} \geq \alpha\}$.
    Prune connections with importance score $s_{ij}^{(l)} < \hat{s}_{p_0 j}^{(l)}$.
  **end for**
**end for**

An important concept in the proposed iPINN strategy is that the pruning method is used to train task-specific subnetworks, but allowing the subnetworks to *naturally* overlap on some connections (see Fig. 2). This way the method provides knowledge sharing between the subnetworks. These overlaps are updated with respect to all tasks that are assigned to a particular connection. Let us denote the loss of each task $\mathcal{D}_j$ as $\mathcal{L}_j = \mathcal{L}(\theta_j;\mathcal{D}_j)$, where $\theta_j$ is the parameter vector for task $\mathcal{D}_j$, $1 \le j \le k$. Then the total loss and its gradient with respect to a parameter $w$ can be written as:

$$\mathcal{L} = \sum_{j=1}^{k} \mathcal{L}_j, \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial w} = \sum_{j=1}^{k} \frac{\partial \mathcal{L}_j}{\partial w} = \sum_{j:\ w \in \mathcal{N}_j} \frac{\partial \mathcal{L}_j}{\partial w}, \tag{10}$$

because if $w \notin \mathcal{N}_j$, then $\frac{\partial \mathcal{L}_j}{\partial w} = 0$.

**Algorithm 2** PINN incremental learning: adding new task $k$

**Require:** neural network $\mathcal{N}$, training datasets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{k-1}$ and $\mathcal{D}_k$, training hyperparameters, pruning hyperparameters ($num\_iters$, $\alpha$).

1: $\mathcal{N}_k \leftarrow \mathcal{N}$ ▷ set full network as a subnetwork
2: Train $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$ on tasks $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$ using Eq. 10. ▷ training step
3: **for** $it = 1, 2, \ldots, num\_iters$ **do** ▷ repeat pruning
4: $\mathcal{N}_k \leftarrow NNrelief(\mathcal{N}_k, \mathcal{D}_k, \alpha)$ ▷ pruning step: Algorithm 1
5: Retrain subnetworks $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$ on tasks $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$ using Eq. 10. ▷ retraining step
6: **end for**

Algorithm 2 includes the pseudocode for iPINNs. For every new task $k$ that enters the network, we first find a corresponding subnetwork $\mathcal{N}_k$ with NNrelief (line 4 of Algorithm 2), then adapt the overlaps between previous subnetworks $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{k-1}$ and a new one $\mathcal{N}_k$ (line 5 of the Algorithm 2). We can prune a (sub)network multiple times (hyperparameter *num_iters*) to achieve a lower sparsity level, however, this is computationally expensive. Therefore we prune every network only once and control the sparsity level with parameter $\alpha$.

***Remark 2*** The pruning strategy allows us to have more flexible variation in parameter sharing because we can keep task-specific parameters within a subnewtork that are not shared with other subnetworks, but we can also keep parameters that are shared among different subnetworks. Task-specific parameters are shown by the red and blue connections in Fig. 2, and they result from pruning the entire network and training free connections (in black) that are unused. The magenta connections in Fig. 2 highlight cases where their parameters are being shared across different tasks, and they result from the pruning algorithm not removing those connections when training for a new task.

The main advantage of the proposed approach is that a neural network learns *all* tasks (equations) that were given during training and not only the last one. This is achieved by constantly replaying old data. Data for previous tasks is easily available by sampling collocation points, which eliminates all issues of data replaying for continual learning problems in computer vision and natural language processing tasks and makes the algorithm well-suited in the context of PINNs. We want to emphasize that iPINN does not need to know how many tasks will be handled overall, and it accesses only those that were considered up to task $k$ inclusive, which distinguishes it from multi-task learning. In the next section, we experimentally show that pretrained parts of the network help to improve the convergence process.

## 5 Numerical experiments

Our findings illustrate the advantage of Algorithm 2 over regular PINNs [20]. The Algorithm allows the network to learn multiple equations (P1) from the same family. Furthermore, by starting with simpler tasks, the network can

**Table 1** Final error and forgetting after all reaction equations are learned

| | Regular PINN [20] | Curriculum training [35] | iPINN (ours) |
|---|---|---|---|
| $\rho = 1$ | | | |
| abs. err | $1.09 \times 10^{-3}$ | $1.09 \times 10^{-3}$ | $\mathbf{1.5 \times 10^{-4}}$ |
| rel. err | 0.263% | 0.263% | **0.039**% |
| $\rho = 2$ | | | |
| abs. err | $1.97 \times 10^{-3}$ | $6.13 \times 10^{-4}$ | $\mathbf{2.5 \times 10^{-4}}$ |
| rel. err | 0.479% | 0.154% | **0.070**% |
| $\rho = 3$ | | | |
| abs. err | $6.72 \times 10^{-3}$ | $1.5 \times 10^{-3}$ | $\mathbf{6.1 \times 10^{-4}}$ |
| rel. err | 2.05% | 0.467% | **0.210**% |
| $\rho = 4$ | | | |
| abs. err | $1.13 \times 10^{-2}$ | $2.89 \times 10^{-3}$ | $\mathbf{1.18 \times 10^{-3}}$ |
| rel. err | 3.68% | 0.98% | **0.458**% |
| $\rho = 5$ | | | |
| abs. err | $5.04 \times 10^{-2}$ | $4.54 \times 10^{-3}$ | $\mathbf{1.91 \times 10^{-3}}$ |
| rel. err | 12.19% | 1.62% | **0.763**% |
| BWT | | | |
| abs. err | N/A | N/A | $-3.8 \times 10^{-4}$ |
| rel. err | N/A | N/A | −0.112% |

Bold values indicate better performance

subsequently learn more complex ones that cannot be learned in isolation.

**Experiments setup** Let us start by examining the proposed algorithms on the convection and reaction equations with periodic boundary conditions (P1). Following the setup in [35], we use a four-layer neural network with 50 neurons per layer. We use 1000 randomly selected collocation points on every time interval between 0 and 1 for $\mathcal{L}_{\mathcal{F}}$. Adam is used as the optimizer [71] with a learning rate of 0.01 and 20,000 epochs to train the model before and after pruning. We divide the learning rate by 3 every 500 epochs in which the loss does not decrease. We repeat our experiments multiple times with different random initializations of the network parameters and show the average values of error. We also compare our approach with curriculum training [35] and use the same total number of training epochs for both curriculum regularization and iPINNs.

To evaluate the performance of the algorithms we compare the final error after the last task. In addition, following continual learning literature [72], we compare backward and forward transfer metrics. Let us denote the test set as $\mathcal{D}^{test} = \{(x^i, t^i, l) : x^i \in [0, 2\pi],\ t^i \in [0, 1],\ l \text{ is the task-ID}\}$ and $N = \#\mathcal{D}^{test}$, the solution of the equation at the point $(x^i, t^i, l)$ as $\mathbf{u}^i_{l,k} = u^i_{l,k}(x^i, t^i)$, and $\hat{\mathbf{u}}^i_{l,k}$ is a prediction of the model at point $(x^i, t^i, l)$ after task $\mathcal{D}_k$ is learned. Relative and absolute errors are denoted as $r_{l,k}$ and $\varepsilon_{l,k}$, respectively, as they are calculated for task $l$ after task $k$ is learned ($l \leq k$).

$$\text{Relative error:} \quad r_{l,k} = \frac{||\mathbf{u}_l - \hat{\mathbf{u}}_{l,k}||_2}{||\mathbf{u}_l||_2} \times 100\%, \tag{11}$$

$$\text{Absolute error:} \quad \varepsilon_{l,k} = \frac{1}{N} \sum_{i=1}^{N} |\mathbf{u}^i_l - \hat{\mathbf{u}}^i_{l,k}|, \tag{12}$$

$$\text{Backward Transfer:} \quad \text{BWT} = \frac{1}{k-1} \sum_{l=1}^{k-1} \varepsilon_{l,k} - \varepsilon_{l,l} \ \text{ or} \tag{13}$$

$$\text{BWT} = \frac{1}{k-1} \sum_{l=1}^{k-1} r_{l,k} - r_{l,l} \tag{14}$$

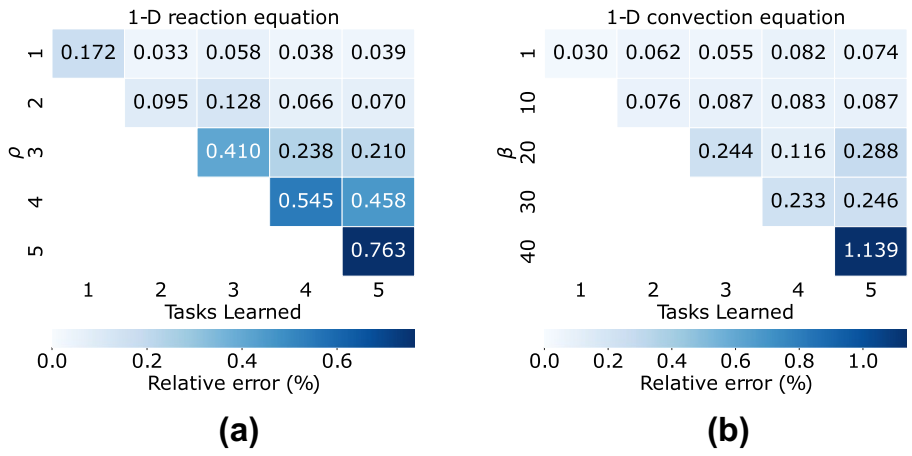



**Fig. 3** Relative error history for reaction equations (**a**) and convection equations (**b**). Every row shows the error after a new task is learned

**Table 2** Final error and forgetting after all convection equations are learned

| | Regular PINN [20] | Curriculum training [35] | iPINN (ours) |
|---|---|---|---|
| $\beta = 1$ | | | |
| abs. err | $\mathbf{2.3 \times 10^{-4}}$ | $\mathbf{2.3 \times 10^{-4}}$ | $4.2 \times 10^{-4}$ |
| rel. err | **0.042**% | **0.042**% | 0.074% |
| $\beta = 10$ | | | |
| abs. err | $1.3 \times 10^{-3}$ | $1.25 \times 10^{-3}$ | $\mathbf{5.0 \times 10^{-4}}$ |
| rel. err | 0.222% | 0.211% | **0.087**% |
| $\beta = 20$ | | | |
| abs. err | $1.9 \times 10^{-3}$ | $\mathbf{1.66 \times 10^{-3}}$ | $\mathbf{1.66 \times 10^{-3}}$ |
| rel. err | 0.339% | 0.298% | **0.288**% |
| $\beta = 30$ | | | |
| abs. err | $2.2 \times 10^{-1}$ | $3.7 \times 10^{-3}$ | $\mathbf{1.44 \times 10^{-3}}$ |
| rel. err | 3.957% | 0.690% | **0.246**% |
| $\beta = 40$ | | | |
| abs. err | $2.3 \times 10^{-1}$ | $2.0 \times 10^{-2}$ | $\mathbf{6.02 \times 10^{-3}}$ |
| rel. err | 37.4% | 3.513% | **1.139**% |
| BWT | | | |
| abs. err | N/A | N/A | $1.8 \times 10^{-4}$ |
| rel. err | N/A | N/A | 0.0280% |

Bold values indicate better performance

## 5.1 Results

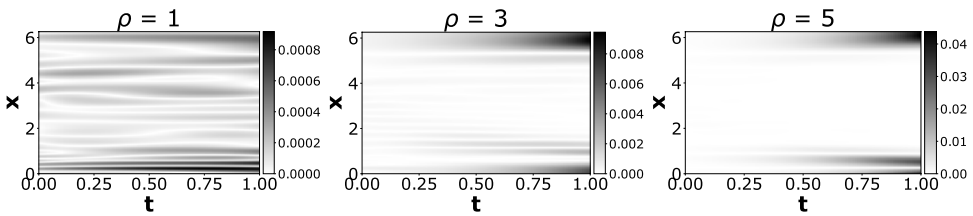
Table 1 presents the results after all reaction equations are learned varying $\rho$ from 1 to 5. Figure 3a shows the error history for every equation after incremental steps. The Table summarizes the performance improvement of iPINNs compared to regular PINNs and curriculum learning [35], exhibiting negligible error for all values of $\rho$, which is especially relevant for cases when $\rho$ is larger. We believe by learning the given PDEs together, we achieve better generalization capabilities due to the synergistic effect of sharing subnetwork parameters. Moreover, iPINNs provide negative BWT which means that previous subnetworks help to learn the following ones.

Similarly, we observe for the convection equation the same learning behaviour. By learning incrementally the sequence of convection equations, we achieve much lower absolute and relative errors for the equations that are more difficult to learn ($\beta = 30, 40$). In Table 2 we show final errors at the end of the training, and Fig. 3b shows the absolute error history for each equation. In this case, we observe some level of forgetting, however, it is insignificant compared to the error values.

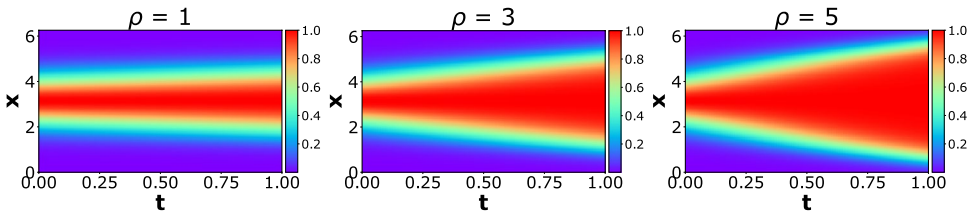
In Figs. 4 and 5, we illustrate the error of iPINNs on convection and reaction equations and the exact solutions for every value of parameter $\beta$ or $\rho$ that were considered. Overall, we see that the neural network learns more complicated tasks more accurately if parts of the network are pretrained with easier tasks. At the same time, iPINNs replay the training data for previous PDEs during training for the new one. There are no additional costs to store or generate input points $(x, t)$ for previous tasks since they can be easily sampled when necessary.

***Remark 3*** The proposed iPINNs may require more training epochs than standard PINNs for one PDE because of the subnetwork creation strategy. However, the algorithm pursues a different goal: provide the ability to learn the solutions sequentially sharing previously learned knowledge.

We also illustrate the effectiveness of the iPINN method by addressing problem P2. We consider the values of $\rho$ and $\nu$ for which a PINN does not have difficulties when learning

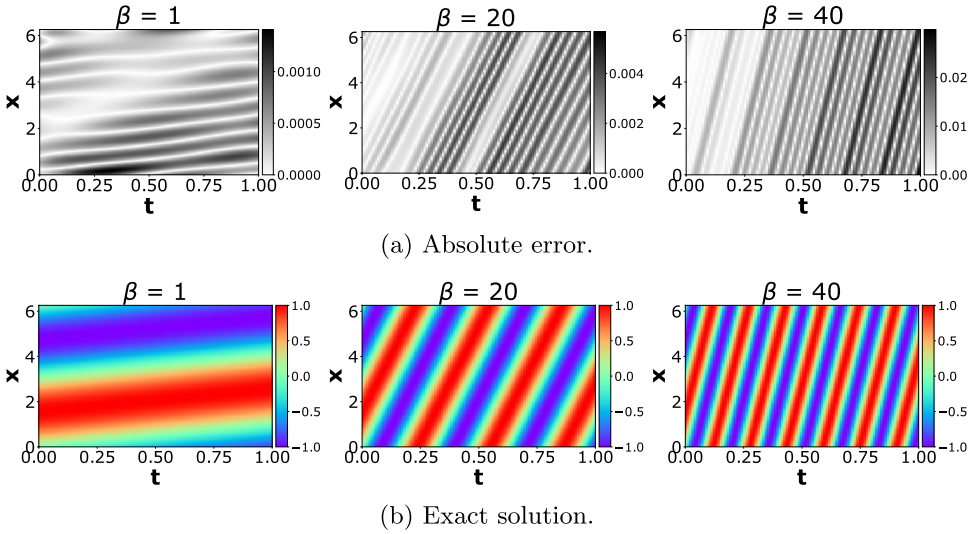



(a) 1-D reaction equation (absolute error).




(b) 1-D reaction equation (exact solution).

**Fig. 4** iPINNs on 1-D reaction equation




(a) Absolute error.

(b) Exact solution.

**Fig. 5** iPINNs on 1-D convection equation

**Table 3** Final error and forgetting for reaction → diffusion → reaction–diffusion

| Parameters | Equation | | Regular PINN [20] | iPINN (ours) |
|---|---|---|---|---|
| $\rho = 3,\ \nu = 5$ | Reaction | abs. err | $6.72 \times 10^{-3}$ | $\mathbf{9.41 \times 10^{-4}}$ |
| | | rel. err | 2.05% | **0.31**% |
| | Diffusion | abs. err | $\mathbf{1.38 \times 10^{-4}}$ | $1.85 \times 10^{-4}$ |
| | | rel. err | **0.05**% | 0.06% |
| | Reaction–diffusion | abs. err | $4.89 \times 10^{-3}$ | $\mathbf{4.10 \times 10^{-3}}$ |
| | | rel. err | 0.80% | **0.68**% |
| $\rho = 4,\ \nu = 4$ | Reaction | abs. err | $1.13 \times 10^{-2}$ | $\mathbf{7.88 \times 10^{-3}}$ |
| | | rel. err | 3.68% | **2.99**% |
| | Diffusion | abs. err | $\mathbf{4.35 \times 10^{-4}}$ | $5.84 \times 10^{-4}$ |
| | | rel. err | **0.16**% | 0.19% |
| | Reaction–diffusion | abs. err | $4.58 \times 10^{-3}$ | $\mathbf{4.42 \times 10^{-3}}$ |
| | | rel. err | 0.70% | **0.67**% |
| $\rho = 4,\ \nu = 5$ | Reaction | abs. err | $5.04 \times 10^{-2}$ | $\mathbf{4.20 \times 10^{-3}}$ |
| | | rel. err | 12.19% | **1.71**% |
| | Diffusion | abs. err | $5.18 \times 10^{-4}$ | $\mathbf{2.30 \times 10^{-4}}$ |
| | | rel. err | 0.18% | **0.08**% |
| | Reaction–diffusion | abs. err | $4.61 \times 10^{-3}$ | $\mathbf{4.58 \times 10^{-3}}$ |
| | | rel. err | 0.69% | **0.68**% |

Bold values indicate better performance

each component of the reaction–diffusion separately. Results obtained when first learning the reaction part (or vice-versa, the diffusion part) are shown in Table 3 (Table 4). The main finding is that the network can learn every equation at least as well as when it is learned independently. In fact, for the reaction equation, the neural network improves significantly the prediction error. Another interesting observation is that the model learns the reaction–diffusion equation with almost the same error, regardless of the order of the tasks. This gives us a hint about the robustness of the algorithm to different task orders in terms of prediction error. In Sect. 6.2, we analyze the percentages of parameters assigned to every subnetwork to illustrate the same conclusion in terms of the number of allocated parameters.

***Remark 4*** In contrast to meta-learning strategies, iPINNs do not need to adapt weights for a new task during testing. One can use the learned model and make a prediction with it.

**Table 4** Final error and forgetting for diffusion → reaction → reaction–diffusion

| Parameters | Equation | | Regular PINN [20] | iPINN (ours) |
|---|---|---|---|---|
| $\rho = 3,\ \nu = 5$ | Diffusion | abs. err | $\mathbf{1.38 \times 10^{-4}}$ | $8.64 \times 10^{-4}$ |
| | | rel. err | **0.05**% | 0.28% |
| | Reaction | abs. err | $6.72 \times 10^{-3}$ | $\mathbf{2.11 \times 10^{-3}}$ |
| | | rel. err | 2.05% | **0.68**% |
| | Reaction–diffusion | abs. err | $4.89 \times 10^{-3}$ | $\mathbf{4.07 \times 10^{-3}}$ |
| | | rel. err | 0.80% | **0.67**% |
| $\rho = 4,\ \nu = 4$ | Diffusion | abs. err | $4.35 \times 10^{-4}$ | $\mathbf{3.45 \times 10^{-4}}$ |
| | | rel. err | 0.16% | **0.12**% |
| | Reaction | abs. err | $1.13 \times 10^{-2}$ | $\mathbf{4.91 \times 10^{-3}}$ |
| | | rel. err | 3.68% | **1.97**% |
| | Reaction–diffusion | abs. err | $4.58 \times 10^{-3}$ | $\mathbf{4.42 \times 10^{-3}}$ |
| | | rel. err | 0.70% | **0.67**% |
| $\rho = 4,\ \nu = 5$ | Diffusion | abs. err | $\mathbf{5.18 \times 10^{-4}}$ | $1.05 \times 10^{-3}$ |
| | | rel. err | **0.18**% | 0.33% |
| | Reaction | abs. err | $5.04 \times 10^{-2}$ | $\mathbf{9.15 \times 10^{-3}}$ |
| | | rel. err | 12.19% | **3.39**% |
| | Reaction–diffusion | abs. err | $4.61 \times 10^{-3}$ | $\mathbf{4.30 \times 10^{-3}}$ |
| | | rel. err | 0.69% | **0.65**% |

Bold values indicate better performance

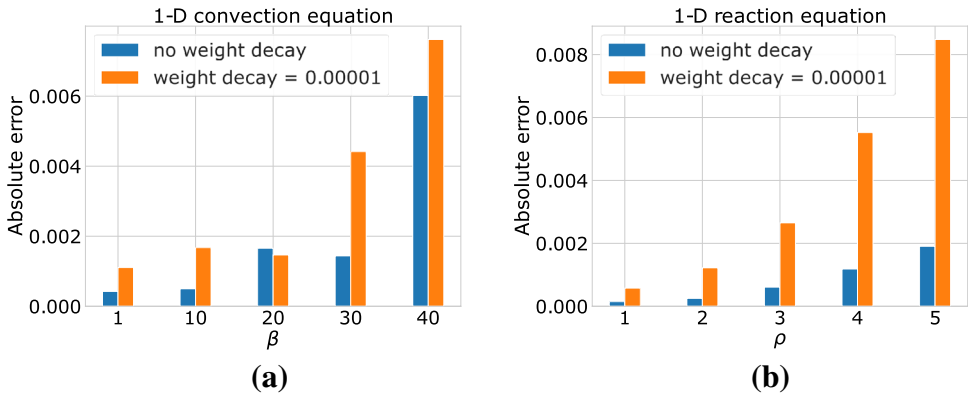



**Fig. 6** Influence of weight decay on the results for reaction (left) and convection (right) equations after all tasks are learned

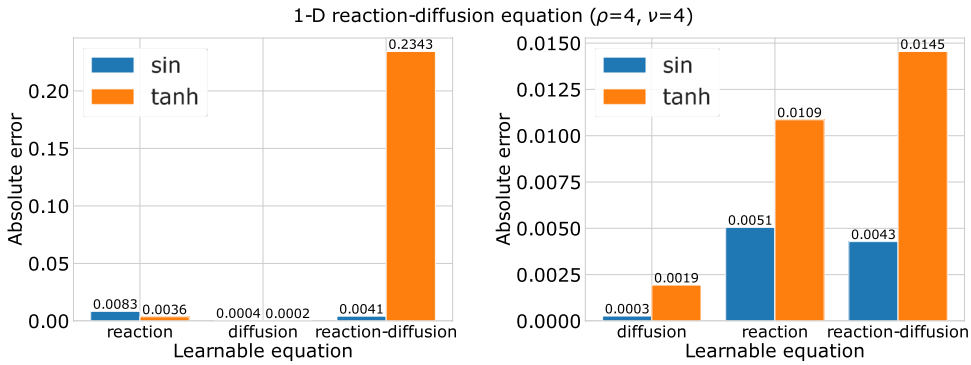



**Fig. 7** Influence of activation function on the results when the reaction learned first (left) and diffusion learned first (right)

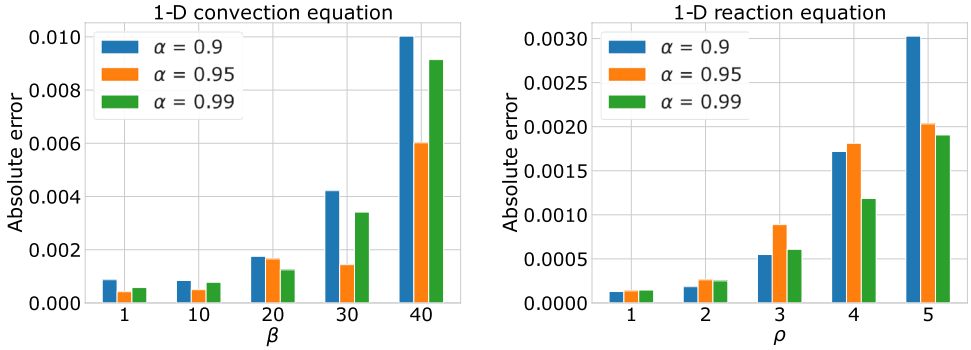



**Fig. 8** iPINNs with different values of pruning parameter $\alpha$

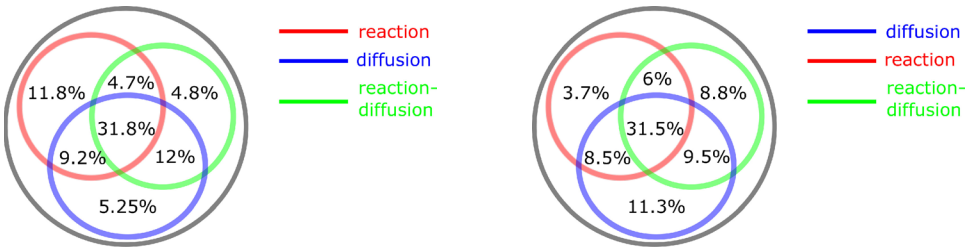



**Fig. 9** Percentage of parameters used for every equation with $\rho = 4,\ \nu = 4$

(a) reaction → diffusion → reaction-diffusion.

(b) diffusion → reaction → reaction-diffusion.

## 6 Additional study

In this section, we provide additional information about the learning procedure of iPINNs . We highlight some important training details such as the presence of regularization and the choice of activation functions. Also, we explore the subnetworks that our approach produces showing the proportion of parameters allocated to each task.

### 6.1 Sensitivity to hyperparameters

Here we illustrate the influence of different training hyperparameters on the performance of iPINNs. First, we compare the results with and without regularization parameter (weight decay). In Fig. 6, it can be observed that the presence of weight decay worsens the prediction error. However, looking at the result it is clear that iPINNs still work if weight decay is present. We can explain the lack of need for weight decay with the fact that many parameters are assigned to multiple tasks and cannot overfit to a particular one. Each subnetwork is also less parameterized than the original network and therefore does not tend to overfit. Thus, weight decay is not necessary and its presence only worsens the result due to the complication of the optimization procedure.

Furthermore, we compare the performance when using `sin` and `tanh` activation functions for two task orderings in Fig. 7. We observe that `sin` works significantly better in both cases. Also, we test `ReLU` activation but it exhibits poor performance in both PDE orderings, as expected. If the reaction is learned first, the absolute errors are 0.4959, 0.2369 and 0.1493. If we start with the diffusion equation and then learn reaction and reaction–diffusion PDEs, the errors are 0.2399, 0.2977 and 0.3003.

In addition, we present how different values of pruning parameter $\alpha$ affect the results. The higher the value of $\alpha$ is, the less the network is pruned. Therefore, if $\alpha = 0.95$ the task-specific subnetworks are sparser than with $\alpha = 0.99$ but less sparse if $\alpha = 0.9$. In Fig. 8, we observe that for the reaction equation, we can prune less and achieve better performance which can be explained by the fact that PDEs in the reaction family are quite similar. Therefore, we can allow the network to have more overlaps to share knowledge between subnetworks. For the case of learning within the same family of convection PDEs, the value of $\alpha = 0.95$ was revealed to be a better option for constructing a sufficiently expressive task-specific subnetwork and frees space for future tasks. Notwithstanding, the performance is good with any reasonable choice of pruning parameter.

### 6.2 Subnetworks analysis

In Fig. 9, we present the portions of the subnetworks that are occupied by each task. We will illustrate this by considering both orders – when the model learns the reaction equation first (Fig. 9a), and when diffusion comes first (Fig. 9b). These results are averaged over 3 different runs for each of the orderings. It is noteworthy that the percentage of parameters occupied by all tasks is very similar for both orderings

(31.8% and 31.5% respectively of all network parameters). On the other hand, the percentages of used parameters for both cases are 79.5% and 79.3%. This means that the total number of trained parameters for the two incremental procedures is the same for both cases, which shows the robustness of the method. Moreover, the network has about 20% of free connections to learn new tasks.

## 7 Conclusion

In this work, we propose an incremental learning approach for PINNs where every task is presented as a new PDE. Our algorithm is based on task-related subnetworks for every task obtained by iterative pruning. To illustrate our idea, we consider two cases when incremental learning is applicable to a sequence of PDEs. In the first case, we consider the family of convection/reaction PDEs, learning them sequentially. In the second example, we consider the reaction–diffusion equation and learn firstly the components of the process, namely reaction and diffusion, and only then the reaction–diffusion equation. Our main goal is to show the possibility of incremental learning for PINNs without significantly forgetting previous tasks. From our numerical experiments, the proposed algorithm can learn all the given tasks, which is not possible with standard PINNs. Importantly, we also show that future tasks are learned better because they can share connections trained from previous tasks, leading to significantly better performance than if these tasks were learned independently. We demonstrate that this stems from the transfer of knowledge occurring between subnetworks that are associated with each task. Interestingly, the model's performance on previous tasks is also improved by learning the following tasks. In essence, iPINNs demonstrate symbiotic training effects between past and future tasks by learning them with a single network composed of dedicated subnetworks for each task that share relevant neuronal connections.

**Author contributions** A.D. wrote the main manuscript text and prepared the figures, A.D. and M.A.B. designed the research, and all authors reviewed the manuscript.

**Data availability** There is no need for sharing data, as the code is made freely available and all results can be replicated using that code.

**Code availability** The code implementation is available at: https://github.com/adekhovich/incremental_PINNs.

## Declarations

**Conflict of interest** The authors have no conflict of interest to declare.

## References


1. Madrazo CF, Heredia I, Lloret L, Lucas JM (2019) Application of a convolutional neural network for image classification for the analysis of collisions in high energy physics. In: EPJ Web of Conferences, vol 214. EDP Sciences, p 06017
2. Bogatskiy A, Anderson B, Offermann J, Roussi M, Miller D, Kondor R (2020) Lorentz group equivariant neural network for particle physics. In: International conference on machine learning. PMLR, pp 992–1002
3. Shlomi J, Battaglia P, Vlimant J-R (2020) Graph neural networks in particle physics. Mach Learn Sci Technol 2(2):021001
4. Khatib O, Ren S, Malof J, Padilla WJ (2022) Learning the physics of all-dielectric metamaterials with deep Lorentz neural networks. Adv Opt Mater 10:2200097
5. Marques G, Agarwal D, Torre Díez I (2020) Automated medical diagnosis of covid-19 through efficientnet convolutional neural network. Appl Soft Comput 96:106691
6. Si T, Bagchi J, Miranda PB (2022) Artificial neural network training using metaheuristics for medical data classification: an experimental study. Expert Syst Appl 193:116423
7. Sarvamangala D, Kulkarni RV (2022) Convolutional neural networks in medical image understanding: a survey. Evol Intel 15(1):1–22
8. Hosaka T (2019) Bankruptcy prediction using imaged financial ratios and convolutional neural networks. Expert Syst Appl 117:287–299
9. Yu P, Yan X (2020) Stock price prediction based on deep neural networks. Neural Comput Appl 32(6):1609–1628
10. Gogas P, Papadimitriou T (2021) Machine learning in economics and finance. Comput Econ 57(1):1–4
11. Wang J, Gan X (2023) Neurodynamics-driven portfolio optimization with targeted performance criteria. Neural Netw 157:404–421
12. Bessa MA, Bostanabad R, Liu Z, Hu A, Apley DW, Brinson C, Chen W, Liu WK (2017) A framework for data-driven analysis of materials under uncertainty: countering the curse of dimensionality. Comput Methods Appl Mech Eng 320:633–667
13. Sosnovik I, Oseledets I (2019) Neural networks for topology optimization. Russ J Numer Anal Math Model 34(4):215–223
14. Chandrasekhar A, Suresh K (2021) Tounn: topology optimization using neural networks. Struct Multidisci Optim 63(3):1135–1149
15. Juan NP, Valdecantos VN (2022) Review of the application of artificial neural networks in ocean engineering. Ocean Eng 259:111947
16. Lee H, Kang IS (1990) Neural algorithm for solving differential equations. J Comput Phys 91(1):110–131
17. Dissanayake M, Phan-Thien N (1994) Neural-network-based approximations for solving partial differential equations. Commun Numer Methods Eng 10(3):195–201
18. Meade AJ Jr, Fernandez AA (1994) Solution of nonlinear ordinary differential equations by feedforward neural networks. Math Comput Model 20(9):19–44
19. Yentis R, Zaghloul M (1996) VLSI implementation of locally connected neural network for solving partial differential equations. IEEE Trans Circuits Syst I Fundam Theory Appl 43(8):687–690
20. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
21. Lagaris IE, Likas A, Fotiadis DI (1997) Artificial neural network methods in quantum mechanics. Comput Phys Commun 104(1–3):1–14

22. Lagaris IE, Likas A, Fotiadis DI (1998) Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 9(5):987–1000
23. Mao Z, Jagtap AD, Karniadakis GE (2020) Physics-informed neural networks for high-speed flows. Comput Methods Appl Mech Eng 360:112789
24. Wessels H, Weißenfels C, Wriggers P (2020) The neural particle method—an updated Lagrangian physics informed neural network for computational fluid dynamics. Comput Methods Appl Mech Eng 368:113127
25. Cai S, Mao Z, Wang Z, Yin M, Karniadakis GE (2022) Physics-informed neural networks (PINNs) for fluid mechanics: a review. Acta Mech Sin 37:1727–1738
26. Chen Y, Lu L, Karniadakis GE, Dal Negro L (2020) Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Opt Express 28(8):11618–11633
27. Lu L, Pestourie R, Yao W, Wang Z, Verdugo F, Johnson SG (2021) Physics-informed neural networks with hard constraints for inverse design. SIAM J Sci Comput 43(6):1105–1132
28. Wiecha PR, Arbouet A, Girard C, Muskens OL (2021) Deep learning in nano-photonics: inverse design and beyond. Photonics Res 9(5):182–200
29. Bai Y, Chaolu T, Bilige S (2022) The application of improved physics-informed neural network (IPINN) method in finance. Nonlinear Dyn 107(4):3655–3667
30. Jagtap AD, Karniadakis GE (2021) Extended physics-informed neural networks (XPINNs): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. In: AAAI Spring Symposium: MLPS
31. Zou Z, Karniadakis GE (2023) L-hydra: multi-head physics-informed neural networks. arXiv preprint arXiv:2301.02152
32. Liu X, Zhang X, Peng W, Zhou W, Yao W (2022) A novel meta-learning initialization method for physics-informed neural networks. Neural Comput Appl 34(17):14511–14534
33. Penwarden M, Zhe S, Narayan A, Kirby RM (2023) A metalearning approach for physics-informed neural networks (PINNs): application to parameterized pdes. J Comput Phys 477:111912
34. Wang S, Teng Y, Perdikaris P (2021) Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM J Sci Comput 43(5):3055–3081
35. Krishnapriyan A, Gholami A, Zhe S, Kirby R, Mahoney MW (2021) Characterizing possible failure modes in physics-informed neural networks. Adv Neural Inf Process Syst 34:26548–26560
36. Rohrhofer FM, Posch S, Gößnitzer C, Geiger BC (2022) Understanding the difficulty of training physics-informed neural networks on dynamical systems. arXiv preprint arXiv:2203.13648
37. Mojgani R, Balajewicz M, Hassanzadeh P (2022) Lagrangian pinns: a causality-conforming solution to failure modes of physics-informed neural networks. arXiv preprint arXiv:2205.02902
38. Wight CL, Zhao J (2020) Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks. arXiv preprint arXiv:2007.04542
39. Castro FM, Marín-Jiménez MJ, Guil N, Schmid C, Alahari K (2018) End-to-end incremental learning. In: Proceedings of the European Conference on Computer Vision (ECCV), pp 233–248
40. Cermelli F, Geraci A, Fontanel D, Caputo B (2022) Modeling missing annotations for incremental learning in object detection. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 3700–3710
41. Kang M, Park J, Han B (2022) Class-incremental learning by knowledge distillation with adaptive feature consolidation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 16071–16080
42. Guo Y, Zhang C, Zhang C, Chen Y (2018) Sparse dnns with improved adversarial robustness. In: Advances in neural information processing systems, vol 31
43. Ahmad S, Scheinkman L (2019) How can we be so dense? The benefits of using highly sparse representations. arXiv preprint arXiv:1903.11257
44. Ye S, Xu K, Liu S, Cheng H, Lambrechts J-H, Zhang H, Zhou A, Ma K, Wang Y, Lin X (2019) Adversarial robustness vs. model compression, or both? In: Proceedings of the IEEE/CVF international conference on computer vision, pp 111–120
45. Liao N, Wang S, Xiang L, Ye N, Shao S, Chu P (2022) Achieving adversarial robustness via sparsity. Mach Learn 111(2):685–711
46. Frankle J, Carbin M (2018) The lottery ticket hypothesis: finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635
47. Mallya A, Lazebnik S (2018) Packnet: adding multiple tasks to a single network by iterative pruning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7765–7773
48. Sokar G, Mocanu DC, Pechenizkiy M (2021) Spacenet: make free space for continual learning. Neurocomputing 439:1–11
49. Sokar G, Mocanu DC, Pechenizkiy M (2022) Avoiding forgetting and allowing forward transfer in continual learning via sparse networks. In: Joint European conference on machine learning and knowledge discovery in databases. Springer
50. LeCun Y, Denker J, Solla S (1989) Optimal brain damage. In: Advances in neural information processing systems, vol 2
51. Hassibi B, Stork D (1992) Second order derivatives for network pruning: optimal brain surgeon. In: Advances in neural information processing system, vol 5
52. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems, vol 28
53. Dong X, Chen S, Pan S (2017) Learning to prune deep neural networks via layer-wise optimal brain surgeon. In: Advances in neural information processing systems, vol 30
54. Dekhovich A, Tax DM, Sluiter MH, Bessa MA (2021) Neural network relief: a pruning algorithm based on neural activity. arXiv preprint arXiv:2109.10795
55. Szandała T (2021) In: Bhoi AK, Mallick PK, Liu C-M, Balas VE (eds) Review and comparison of commonly used activation functions for deep neural networks. Springer, Singapore, pp 203–224
56. Jagtap AD, Karniadakis GE (2022) How important are activation functions in regression and classification? A survey, performance comparison, and future directions. arXiv preprint arXiv:2209.02681
57. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: International Conference on Machine Learning
58. Sitzmann V, Martel J, Bergman A, Lindell D, Wetzstein G (2020) Implicit neural representations with periodic activation functions. Adv Neural Inf Process Syst 33:7462–7473
59. Bengio Y (2012) Deep learning of representations for unsupervised and transfer learning. In: Proceedings of ICML workshop on unsupervised and transfer learning. JMLR Workshop and Conference Proceedings, pp 17–36
60. Tan C, Sun F, Kong T, Zhang W, Yang C, Liu C (2018) A survey on deep transfer learning. In: International conference on artificial neural networks. Springer, pp 270–279

61. Ruder S, Peters ME, Swayamdipta S, Wolf T (2019) Transfer learning in natural language processing. In: Proceedings of the 2019 conference of the North American Chapter of the Association for Computational Linguistics: tutorials, pp 15–18
62. Houlsby N, Giurgiu A, Jastrzebski S, Morrone B, De Laroussilhe Q, Gesmundo A, Attariyan M, Gelly S (2019) Parameter-efficient transfer learning for nlp. In: International conference on machine learning. PMLR, pp 2790–2799
63. Goswami S, Anitescu C, Chakraborty S, Rabczuk T (2020) Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. Theor Appl Fract Mech 106:102447
64. Niaki SA, Haghighat E, Campbell T, Poursartip A, Vaziri R (2021) Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture. Comput Methods Appl Mech Eng 384:113959
65. Chakraborty S (2021) Transfer learning based multi-fidelity physics informed deep neural network. J Comput Phys 426:109942
66. Xu C, Cao BT, Yuan Y, Meschke G (2023) Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios. Comput Methods Appl Mech Eng 405:115852
67. Chen X, Gong C, Wan Q, Deng L, Wan Y, Liu Y, Chen B, Liu J (2021) Transfer learning for deep neural network-based partial differential equations solving. Adv Aerodyn 3(1):1–14
68. Bengio Y, Louradour J, Collobert R, Weston J (2009) Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning, pp 41–48
69. Krizhevsky A (2009) Learning multiple layers of features from tiny images. Master's thesis, University of Tront
70. Le Y, Yang X (2015) Tiny imagenet visual recognition challenge. CS 231N 7(7):3
71. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
72. Lopez-Paz D, Ranzato M (2017) Gradient episodic memory for continual learning. In: Advances in neural information processing systems, vol 30