Journal of Computational Physics 490 (2023) 112291

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Adaptive transfer learning for PINN

Yang Liu [a,b], Wen Liu [a,*], Xunshi Yan [c], Shuaiqi Guo [a,b], Chen-an Zhang [a]

[a] State Key Laboratory of High Temperature Gas Dynamics, Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China
[b] School of Engineering Science, University of Chinese Academy of Sciences, Beijing 100049, China
[c] Institute of Nuclear and New Energy Technology, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 10 November 2022
Received in revised form 8 May 2023
Accepted 7 June 2023
Available online 15 June 2023

Keywords:
Physical-informed neural networks
Transfer learning
Adaptive learning
Non-convex problem
The minimum energy path

ABSTRACT

Physics-informed neural networks (PINNs) have shown great potential in solving computational physics problems with sparse, noisy, unstructured, and multi-fidelity data. However, the training of PINN remains a challenge, and PINN is not robust to deal with some complex problems, such as the sharp local gradient in broad computational domains, etc. Transfer learning techniques can provide fast and accurate training for PINN through intelligent initialization, but the previous researches are much less effective when dealing with transfer learning cases with a large range of parameter variation, which also suffers from the same drawbacks. This manuscript develops the concept of the minimum energy path for PINN and proposes an adaptive transfer learning for PINN (AtPINN). The Partial Differential Equations (PDEs) parameters are initialized by the source parameters and updated adaptively to the target parameters during the training process, which can guide the optimization of PINN from the source to the target task. This process is essentially performed along a designed low-loss path, which is no barrier in the energy landscape of neural networks. Consequently, the stability of the training process is guaranteed. AtPINN is utilized to achieve transfer learning cases with a large range of parameter variation for solving five complex problems. The results demonstrate that AtPINN has promising potential for extending the application of PINN. Besides, three transfer learning cases with different ranges of parameter variation are analyzed through visualization. Furthermore, results also show that the idea of adaptive transfer learning can be a particular optimization strategy to directly solve problems without intelligent initialization.

1. Introduction

Deep learning has had a revolutionary impact in several scientific disciplines, such as computer vision (CV) [1] and natural language processing (NLP) [2]. With the dramatic capability of nonlinear modeling, deep learning has attracted tremendous attention in recent years in the field of computational mechanics [3,4]. However, typical deep learning is constructed as a black-box surrogate model and usually requires a large amount of labeled data, which are often unavailable in many scientific applications [5,6].

Recently, Raissi et al. [7] proposed physics-informed neural networks (PINNs) that known partial differential equations (PDEs) are utilized to constrain (or even drive) the training of neural networks with relatively small amounts of data. The

* Corresponding author.
E-mail addresses: liuyang2@imech.ac.cn (Y. Liu), lw@imech.ac.cn (W. Liu), yanxs@tsinghua.edu.cn (X. Yan), guoshuaiqi@imech.ac.cn (S. Guo), zhch_a@imech.ac.cn (C.-a. Zhang).

https://doi.org/10.1016/j.jcp.2023.112291
0021-9991/

PDEs' residual, initial condition, and boundary condition are incorporated into the loss function of neural networks. In this set, the training of neural networks is driven by minimizing the residual of the governing equations. PINN provides a mesh-free algorithm that exploits automatic differentiation (AD) [8] to represent the differential operators. Besides, PINN has shown the potential in solving inverse problems with observation data [9]. Thanks to the approximation capabilities, PINN has already produced a series of noticeable results on a range of problems in computational science and engineering [10–14]. Despite these advantages, it remains a challenge to train accurate and fast PINN, which is associated with the minimization of the loss function [15]. The loss function is a high-dimensional and non-convex function containing multiple terms that compete with each other during the training process [16,17]. Consequently, the training process may not be sufficiently stable and PINN is not robust to deal with some complex problems, such as sharp local gradient [18] and high-frequency problems [19] in broad computational domains, etc.

Transfer learning is motivated by the fact that human beings can intelligently apply knowledge learned previously to solve a new problem with a better solution [20,21]. It has been used to avoid this non-convex property to obtain an accurate and fast optimization for PINN [22–27]. The transfer learning of PINN aims to transfer the learned weights from the PINN described by the source parameters (source task) to another PINN described by the target parameters (target task). Several works have applied transfer learning for PINN to reduce computational cost and enhance accuracy, such as the incompressible flow modeling [22], the regimes flow modeling [23], the thermochemical curing process modeling [24], the phase-field modeling of fracture [25], linear and non-linear elasticity problems [26], multi-fidelity physical modeling [27], etc. Nevertheless, existing works only focus on transfer learning cases with a small range of parameter variation between the source and target tasks, which means that the two tasks are strongly related or similar. When the range of parameter variation becomes large, it remains an issue whether it is possible to transfer the source task to the target task, which depends on the similarity of the PDE solutions under two parameters. In most cases, the large range of parameter variation means that the target task is so different from the source task that transfer learning is hard to implement. In such cases, transfer learning can be considered as a re-training process that mainly focuses on knowledge transfer at the feature level. Therefore, it may suffer from the same drawbacks as PINN.

Although the mean squared error (MSE) defines a convex form of the loss function, over-parameterized neural networks still face the problem of non-convex optimization [28]. Some studies analyzed the energy landscape of the loss function and tried to design an optimization path in parameter space to avoid this non-convex property [29,30]. The study of [31] and [32] empirically concluded that loss minima are not isolated points but essentially form a connected manifold and there exists a continuous minimum energy path to connect two minima. Sagun et al. [33]. constructed flat linear paths between close minima. Draxler et al. [31]. developed an automated nudged elastic band (AutoNEB) method to construct a flat path between arbitrary minima. However, it remains challenging to comprehensively understand the loss landscape and efficiently design the optimization path due to the huge parameter space [34]. Note that the transfer learning tasks of PINN are described by the PDEs' parameters with an explicit physical definition. The PDEs' parameters can be treated as the trainable weights of PINN, making it easier to construct such an optimization path to guide the training of PINN by fine-tuning PDEs' parameters [35,36].

Inspired by this idea, we develop a feasible minimum energy path for PINN and propose an adaptive transfer learning for PINN (AtPINN). The PDEs' parameters are adaptively updated from the source to the target parameters, which is utilized to construct a low-loss path in the parameter space to guide the optimization of PINN from the source to the target task. During this process, the loss is always kept at a low level, and correspondingly, the solution is very close to the physical solution, which guarantees the stability of the training process. Besides, the gradient behaviors of different loss terms are analyzed to determine the principle of the weight assignment, and the normalization is utilized to extend AtPINN to accommodate cases with multiple parameters. To illustrate the potential of the proposed method, AtPINN is exploited to achieve the transfer learning cases with a large range of parameter variation for solving five complex problems. Furthermore, the transfer learning cases with three ranges of parameter variation are discussed and the optimization process of AtPINN is also analyzed through visualization.

The present study is arranged as follows: Section 2 introduces the background and the transfer learning applications for PINN. Section 3 describes details about the proposed method. Five problems are studied in Section 4 and the results demonstrate the performance of the proposed method. Finally, the conclusion and remarks are presented in Section 5.

2. Background

2.1. Physics-informed neural networks

Raissi et al. [7] proposed the PINN concept for solving the forward and inverse problem of the partial differential equations (PDEs) by embedding the PDEs' residual, initial condition, and boundary condition into the loss function of neural networks. In this set, the problem of solving PDEs is converted into the optimization problem of neural networks. In this section, the classical PINN concept will be reviewed.

Consider a scalar function $u(\boldsymbol{x}, t)$ on the domain $\Omega \times [0, \infty)$ with the boundary $\partial\Omega$, where $\Omega \subset \mathbb{R}^d$. $u(\boldsymbol{x}, t)$ satisfies the following PDEs:

$$\mathcal{F}(\boldsymbol{x}, t; u, \partial_{\boldsymbol{x}} u, \partial_t u, \cdots, \boldsymbol{\lambda}) = 0, \ \forall (\boldsymbol{x}, t) \in \mathcal{U} \tag{1a}$$

$$\mathcal{I}(\boldsymbol{x}, t_0, h; u, \partial_t u, \cdots, \boldsymbol{\lambda}) = 0,\ \forall (\boldsymbol{x}, t) \in \Gamma \tag{1b}$$

$$\mathcal{B}(\boldsymbol{x}, t, g; u, \partial_{\boldsymbol{x}} u, \cdots, \boldsymbol{\lambda}) = 0,\ \forall (\boldsymbol{x}, t) \in \partial\mathcal{U} \tag{1c}$$

where $\mathcal{F}$ contains a series of the differential operators (i.e., $[\partial_t, \partial_{\boldsymbol{x}}, ...]$), which can represent the residual of the PDEs, $\boldsymbol{\lambda}$ is the PDEs' parameter vector, $\mathcal{I}$ is the residual form of the initial condition containing a function $h(\boldsymbol{x}, t)$ and $\mathcal{B}$ is the residual form of the boundary condition containing a function $g(\boldsymbol{x}, t)$. $\mathcal{U} = \{(\boldsymbol{x}, t) | \boldsymbol{x} \in \Omega, t = [0, T]\}$, $\partial\mathcal{U} = \{(\boldsymbol{x}, t) | \boldsymbol{x} \in \partial\Omega, t = [0, T]\}$, and $\Gamma = \{(\boldsymbol{x}, t) | \boldsymbol{x} \in \partial\Omega, t = 0\}$.

In the classic PINN concept, fully-connected feed-forward neural networks are employed. The space coordinate $\boldsymbol{x}$ and time t are usually taken as the inputs, and the outputs $\hat{u}(\boldsymbol{x}, t)$ are used to approximate the true solution $u(\boldsymbol{x}, t)$ of the PDEs. The differential operators are calculated by AD, and then the PDEs' residual, initial condition, and boundary condition are embedding into the loss function of neural networks:

$$\mathcal{L}(\boldsymbol{\theta}) = w_{\mathcal{F}}\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + w_{\mathcal{I}}\mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}) + w_{\mathcal{B}}\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) \tag{2}$$

where $\boldsymbol{\theta}$ denotes the trainable weights of neural networks, $w_{\mathcal{F}}$, $w_{\mathcal{I}}$, and $w_{\mathcal{B}}$ are the weights for different loss terms, and $\mathcal{L}_{\mathcal{F}}$, $\mathcal{L}_{\mathcal{I}}$, and $\mathcal{L}_{\mathcal{B}}$ are the loss functions corresponding to PDEs, initial condition, and boundary condition, respectively:

$$\mathcal{L}_{\mathcal{F}} = \frac{1}{N_{\mathcal{F}}} \sum_{i=1}^{N_{\mathcal{F}}} \|\mathcal{F}(\boldsymbol{x}^{(i)}, t^{(i)}; \hat{u})\|^2 \tag{3a}$$

$$\mathcal{L}_{\mathcal{I}} = \frac{1}{N_{\mathcal{I}}} \sum_{i=1}^{N_{\mathcal{I}}} \|\mathcal{I}(\boldsymbol{x}^{(i)}, t^{(i)}, h^{(i)}; \hat{u})\|^2 \tag{3b}$$

$$\mathcal{L}_{\mathcal{B}} = \frac{1}{N_{\mathcal{B}}} \sum_{i=1}^{N_{\mathcal{B}}} \|\mathcal{B}(\boldsymbol{x}^{(i)}, t^{(i)}, g^{(i)}; \hat{u})\|^2 \tag{3c}$$

where $\{\boldsymbol{x}^{(i)}, t^{(i)}\}_{i=1}^{N_{\mathcal{F}}}$, $\{\boldsymbol{x}^{(i)}, t^{(i)}, h^{(i)}\}_{i=1}^{N_{\mathcal{I}}}$, and $\{\boldsymbol{x}^{(i)}, t^{(i)}, g^{(i)}\}_{i=1}^{N_{\mathcal{B}}}$ are the set of PDEs' residual points in $\mathcal{U}$, Γ, and $\partial\mathcal{U}$, and $N_{\mathcal{F}}$, $N_{\mathcal{I}}$, and $N_{\mathcal{B}}$ donate the number of sampling points. In this manuscript, the loss function of PINN is denoted as $\mathcal{L}_{PINN}(\boldsymbol{\theta})$. PINN can be trained by gradient descent algorithms (such as: Adam [37], SGD [38], and LBGFS [39] algorithms) to minimize the loss function.

PINN is a mesh-free simulation algorithm and shows promising potential in solving many computational physics problems and multi-fidelity data fusion. However, it is still a challenge to train PINN with fast convergence and high accuracy. This challenge is associated with loss functions that contain multiple loss terms and are high-dimensional non-convex functions. The complex loss functions lead to a difficult training process that is too much costly and time-consuming. In addition, PINN models often involve training large-scale neural networks in an over-parameterized regime, for example, by specifying a DNN that has more complexity than the problem requires. If the problems are too complex or the sampling points are not dense enough, convergence to the global minimum cannot be guaranteed and PINN is susceptible to obtaining an inaccurate or even unphysical solution.

2.2. Transfer learning for PINN

In the context of transfer learning, knowledge acquired by the source task that has been trained is transferred to a related, but not identical target task. Transfer learning aims to apply knowledge learned previously to help the solving of a new problem, which is usually performed for a target task with less data in the field of computer vision (CV) and natural language processing (NLP).

Transfer learning techniques have been used to reduce computational costs and enhance accuracy for PINN. In this manuscript, we name the transfer learning method for PINN as tPINN. Fig. 1 presents the basic idea of tPINN. The source task is the PINN (described by $\boldsymbol{\lambda}_s$) that has been trained and knowledge is a set of weights $\boldsymbol{\theta}_s$. If a target task described by the $\boldsymbol{\lambda}_t$ is determined to be related to the source task by human beings, the stored weights $\boldsymbol{\theta}_s$ are provided as a smart initialization for the new task. With this strategy, PINN is trained again to obtain the weights $\boldsymbol{\theta}_t$ and converge much faster on each new problem, resulting in a significant reduction in overall training time. This process may also be interpreted as a two-stage training process. The first stage is preparing a source PINN and the second stage is solving target PINN with the intelligent initialization. The loss function of tPINN is same as PINN, namely $\mathcal{L}_{tPINN}(\boldsymbol{\theta}) = \mathcal{L}_{PINN}(\boldsymbol{\theta})$.

Transfer learning is a feasible solution to alleviate the costly and time-consuming training of PINN. Yet, current studies only focus on problems where the PDEs' parameters of the source task are very close to the parameters of the target task, implying a small range of parameter variation. In such cases, the target task is strongly correlated with the source task, and transfer learning techniques are very well suited to solving these problems. For the transfer learning case with a large range of parameter variation, whether it is possible to transfer the source to the target task remains an issue, which depends on the similarity of their solutions. In many cases, the differences between the solutions of source and target parameters are significant enough that implementing transfer learning can be challenging. For example, the Reynolds number exists in

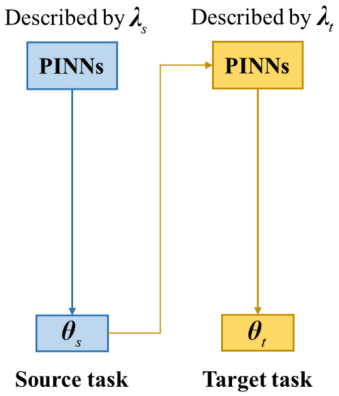


Fig. 1. The basic idea of tPINN. The source PINN (top left) described by $\boldsymbol{\lambda}_s$ that has been trained, and knowledge $\boldsymbol{\theta}_s$ is stored. A target PINN (top right) described by $\boldsymbol{\lambda}_t$ is initialized by a reference weights $\boldsymbol{\theta}_s$, and is trained to get the weights $\boldsymbol{\theta}_t$.

the high-order terms of the Navier-Stocks equations, and a large range of parameter variation often implies a significant change in the solution. Thus, it is less effective or even impractical to directly transfer a PINN describing the low Reynolds number flow to another PINN describing the high Reynolds number flow. Such an example widely exists in most problems of practical interest. Moreover, transfer learning is motivated by human learners, who only focus on knowledge transfer at the feature level and unconsciously recognize two similar tasks to apply experience from previous problems to new ones. There is no clear threshold to tell human learners whether the source task can be successfully transferred to the target task or not, although the parameter variation between the two tasks has been known.

3. Proposed method: adaptive transfer learning for PINN

3.1. Overview of the proposed method

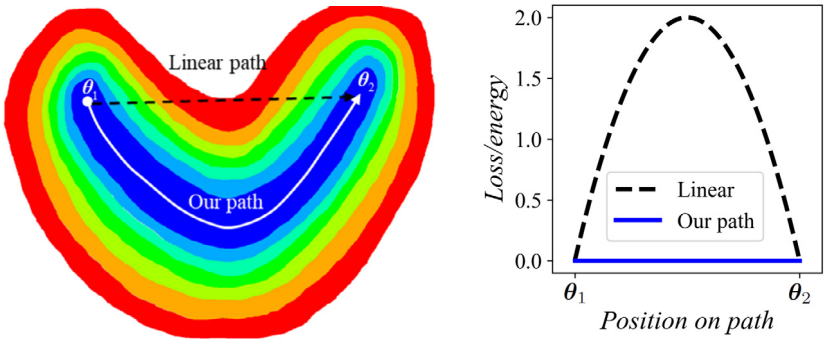
In this manuscript, we propose an adaptive transfer learning method with low loss for PINN inspired by the concept of the minimum energy path. The PDEs' parameters are adaptively updated from the source to the target parameters to guide the optimization of PINN from the source to the target task. During the training process, the loss is kept at a low level so that a low-loss path is constructed in the parameter space. Along the constructed path, the PINN solution is very close to the physical solution, which ensures the stability of the training process.

First, the concept of the loss landscape of neural networks and the minimum energy path for neural networks is reviewed and a feasible minimum energy path for PINN is proposed. In the parameter space, the path is described by the PDEs' parameters that can be treated as the trainable weights with an explicit physical definition. Second, the loss function of transfer learning is redesigned according to the minimum energy path. Based on intelligent initialization, the weights in PINN are fine-tuned parameters to gradually transfer the source to the target task, which essentially proceeds along the designed path. Unlike previous transfer learning studies focusing on knowledge transfer at the feature level, this manuscript is concerned with the fact that the PINN loss is always kept at a low level, which can effectively ensure the stability of PINN. Finally, to enable the proposed method applicable to different scenarios, the principle of weight assignment is determined by analyzing the gradient behaviors of different loss terms. In addition, normalization is leveraged to extend the proposed method to accommodate the scenarios with multiple parameters.

3.2. The loss landscape of neural networks

The loss landscape of neural networks refers to the graphical representation of the loss function over its high-dimensional parameter space. The loss function measures the discrepancy between the predicted output and the actual output, and the parameter space includes all possible values for the weights and biases of the network.

In over-parameterized networks, the number of trainable parameters (weights and biases) exceeds the number of training samples, thus enabling the network to effectively fit the training data. However, non-convexity associated with such over-parameterization networks may lead to complicated and strange optimization landscapes, making it challenging for optimization algorithms to find the global minimum. Research [40] has demonstrated that the loss landscape of over-parameterized neural networks is typically characterized by numerous wide valleys or flat regions. Recent studies [31,33] have revealed that deep neural networks possess a large number of global minima, all of which achieve comparable levels of training error. These can facilitate efficient training and optimization for neural networks. Therefore, it is of great interest to understand the loss landscape of neural networks for developing more efficient optimization strategies.



(a) A slice in the parameter space (b) Losses along different paths

Fig. 2. A schematic of a slice through the high dimensional parameter space of neural networks.

3.3. The minimum energy path

According to literature [31], neural network loss minima are not isolated points in parameter space, but essentially form a connected manifold. Draxler et al. [31] and Garipov et al. [33] empirically found that two minima can be connected by the minimum energy path (or equal-value path), along which the losses are very close to minima. Here, the concept of the minimum energy path will be reviewed.

The loss of neural networks depends on the architecture, the training data set, and the network weights. Keeping the architecture and the training set fixed, the PINN loss function $\mathcal{L}(\boldsymbol{\theta})$ obtains the minima on two weights $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. There exists a continuous equal-value path $\boldsymbol{p}^*$ in the parameter space that connects $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ with the lowest maximum loss:

$$\boldsymbol{p}^*(\boldsymbol{\theta}) = \underset{\boldsymbol{p}\ from\ \boldsymbol{\theta}_1\ to\ \boldsymbol{\theta}_2}{\arg\min} \left\{ \max_{\boldsymbol{\theta}\in\boldsymbol{p}} \mathcal{L}(\boldsymbol{\theta}) \right\} \tag{4}$$

Such a lowest path $\boldsymbol{p}^*$ is called the minimum energy path. Fig. 2 exhibits a schematic of a slice through the high dimensional parameter space of neural networks. The plane is spanned by the two minima $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ and two different paths are defined from the $\boldsymbol{\theta}_1$ to $\boldsymbol{\theta}_2$. Although the linear path is the shortest path connecting two minima, it must pass through the region with high loss/energy, just like a barrier between two minima. Losses along the minimum energy path can be viewed as the minima, and the loss/energy is essentially flat.

The solution of the minimum energy path can be transformed into an optimization problem. To make this problem tractable, the loss function must be sufficiently smooth, e.g., containing no jumps along the path. For most deep neural networks with ReLU or Leaky ReLU activation function, the derivative is discontinuous, which usually requires sampling densely to overcome this problem. To find such the path $\boldsymbol{p}^*$, a chain of $N+2$ pivots $\boldsymbol{p}_i$ (a series of weight sets $\boldsymbol{\theta}_i$) for $i = 0, \ldots, N+1$ is connected via the coefficient k. The initial and final pivots are fixed to the minima $\boldsymbol{p}_0 = \boldsymbol{\theta}_1$ and $\boldsymbol{p}_{N+1} = \boldsymbol{\theta}_2$, respectively. Using the gradient descent to minimize the following energy function:

$$E(\boldsymbol{p}) = \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{p}_i) + \sum_{i=1}^{N} \frac{1}{2} k \|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|^2 \tag{5}$$

where k is the connection coefficient.

The solution of this energy formulation relies on the number of pivots and the coefficient k [41]. If the number is too small or if k is samll, the distances between two adjacent pivots become large, and there is often a high-energy region between them where the sampling should be dense enough. Conversely, if the number is too large or if k is large, the energy grows quadratically with the total length of the path, and solving this problem becomes costly and time-consuming. Note that in practice, it is hard to find the exact minimum around each pivot, and thus the loss $\mathcal{L}(\boldsymbol{p}_i)$ often only needs to be very close to the minima along the path.

3.4. A minimum energy path for PINN

The PDEs' parameters, such as Reynolds number and Mach number, are embedded into the loss function of neural networks as well as the PDE residual, initial condition, and boundary condition. In previous investigations, these parameters were usually considered as constants for solving a specific problem. In this manuscript, the PDEs' parameters are treated as a series of trainable weights rather than constants to obtain PINN solutions under different parameters. This change does not affect the smoothness of the loss function because the solution varies continuously with the PDEs' parameters for problems we are investigating in this study. Moreover, the losses and derivatives of PINN are sufficiently smooth because of the Tanh

activation function. These properties make it easier to construct such a minimum energy path compared with the previous studies.

Suppose the neural network weights $\boldsymbol{\theta} \in V_1$ and the PDEs' parameters $\boldsymbol{\lambda} \in V_2$, where V_1 and V_2 are two different parameter space and $V_1 \cap V_2 = \{\mathbf{0}\}$. We write the direct sum of V_1 and V_2:

$$V = V_1 \oplus V_2 \tag{6}$$

where V is a new parameter space consisting of V_1 and V_2, and $\dim(V) = \dim(V_1) + \dim(V_2)$. For every $\boldsymbol{\Theta} \in V$, there exist unique $\boldsymbol{\theta} \in V_1$ and $\boldsymbol{\lambda} \in V_2$, such that $\boldsymbol{\Theta} = \boldsymbol{\theta} + \boldsymbol{\lambda}$.

Similar to section 3.3, $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are two minima of the PINN loss function, and the corresponding PDEs' parameters are $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$, respectively. For $\boldsymbol{\Theta}_1 = \boldsymbol{\theta}_1 + \boldsymbol{\lambda}_1$ and $\boldsymbol{\Theta}_2 = \boldsymbol{\theta}_2 + \boldsymbol{\lambda}_2$, the minimum energy path and the energy function in the parameter space of V can be defined:

$$\boldsymbol{p}^*(\boldsymbol{\Theta}) = \underset{\boldsymbol{p}\ from\ \boldsymbol{\Theta}_1\ to\ \boldsymbol{\Theta}_2}{\arg\min} \left\{ \max_{\boldsymbol{\Theta} \in \boldsymbol{p}} \mathcal{L}(\boldsymbol{\Theta}) \right\} \tag{7a}$$

$$E(\boldsymbol{p}) = \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{\Theta}_i) + \sum_{i=1}^{N} \frac{1}{2} k \|\boldsymbol{\Theta}_{i+1} - \boldsymbol{\Theta}_i\|^2 \tag{7b}$$

the goal is to find a continuous path $\boldsymbol{p}^*$ from $\boldsymbol{\Theta}_1$ to $\boldsymbol{\Theta}_2$ with a lowest maximum loss in the new parameter space.

It remains challenging to efficiently solve equation (7b) from $\boldsymbol{\Theta}_1$ to $\boldsymbol{\Theta}_2$ due to the huge parameter space of V. In the context of tPINN, the transfer learning process can be described by the PDEs' parameters that are the trainable weights in the parameter space of V_2. Based on this understanding, we define a feasible minimum energy path for PINN that is guided by the PDEs' parameters. Different from equations (7a) and (7b), this involves the inverse problem of solving the minimum energy path, i.e., knowing the weights of source task $\boldsymbol{\Theta}_1 = \boldsymbol{\theta}_1 + \boldsymbol{\lambda}_1$ and the minimum energy path $\boldsymbol{p}^*$ described by $\boldsymbol{\lambda}$ ($\boldsymbol{\lambda}_1 \rightarrow \boldsymbol{\lambda}_2$), solving $\boldsymbol{\theta}_2$. Thus, the inverse problem of the minimum energy path for PINN is defined:

$$\boldsymbol{p}^*(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \underset{\boldsymbol{p}\ from\ \boldsymbol{\lambda}_1\ to\ \boldsymbol{\lambda}_2}{\arg\min} \left\{ \max_{\boldsymbol{\lambda} \in \boldsymbol{p}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \right\} \tag{8}$$

where $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ are the known PDEs' parameters. $\boldsymbol{\lambda}$ is initialized with $\boldsymbol{\lambda}_1$ and gradually fine-tuned towards $\boldsymbol{\lambda}_2$, while leaving the weights $\boldsymbol{\theta}$ free to adapt to this fine-tuning to still minimize the PINN loss $\mathcal{L}(\boldsymbol{\theta})$.

Note that, the previous studies [29,33] tried to find the minimum energy path in huge parameter space of V, such as neural networks with dozens or hundreds of neurons/channels per layer, which is costly and time-consuming. Compared with the previous studies, the advantage of equation (8) is that the PDEs' parameters with clear physical meaning, describe the minimum energy path within low dimensional parameter space of V_2, which defines the minimum energy path that is easy to solve for PINN.

The solution of equation (8) can be transformed into an optimization process. Along the minimum energy path, $N+2$ pivots $\boldsymbol{p}_i$ ($i = 0, \ldots, N+1$, $\boldsymbol{p}_0 = \boldsymbol{\lambda}_1$ and $\boldsymbol{p}_N = \boldsymbol{\lambda}_2$) on the path can be found. For two adjacent pivots i and $i+1$, the energy function is expressed:

$$E(\boldsymbol{p}_i) = \mathcal{L}(\boldsymbol{\theta}_i, \boldsymbol{\lambda}_i) + \frac{1}{2} k \|\boldsymbol{\lambda}_{i+1} - \boldsymbol{\lambda}_i\|^2 \tag{9}$$

To understand this path intuitively, assume PINN has been trained, and a weight set with global minimum loss has been identified. Now if the PDEs' parameters are perturbed by a small constant, but leave the PINN's weights free to adapt to this change to still minimize the PINN loss. It can be argued that the PINN's weights can "make up" the perturbation by adjusting somewhat. This concept is very similar to transfer learning cases in that they both start from intelligence initialization and optimize the network weights to obtain a target model. However, previous transfer learning cases for PINN only focus on knowledge transfer at the feature level and cannot achieve transfer learning cases with a large range of parameter variations. The minimum energy path for PINN is to find an optimization path with low loss, thus ensuring the stability of the training process.

3.5. Adaptive transfer learning for PINN

Previous transfer learning studies for PINN only focus on knowledge transfer at the feature level and are limited to cases with a small range of parameter variation. Once the range of parameter variation becomes large, it also suffers from the drawbacks of the difficult optimization because of the high-dimensional non-convex loss function. Equations (8) and (9) introduce the minimum energy path for PINN that is utilized to overcome the above limitations.

Based on the concept of the minimum energy path, this manuscript proposes an adaptive transfer learning for PINN (AtPINN). As shown in Fig. 3, a series of pivots $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_N$ between $\boldsymbol{\lambda}_s$ and $\boldsymbol{\lambda}_t$ are chosen ($\boldsymbol{\lambda}_s = \boldsymbol{\lambda}_0$, $\boldsymbol{\lambda}_t = \boldsymbol{\lambda}_{N+1}$). Due to the small parameter distance between adjacent pivots $\boldsymbol{\lambda}_i$ and $\boldsymbol{\lambda}_{i+1}$, it is easy to transfer the PINN described by $\boldsymbol{\lambda}_i$ to the

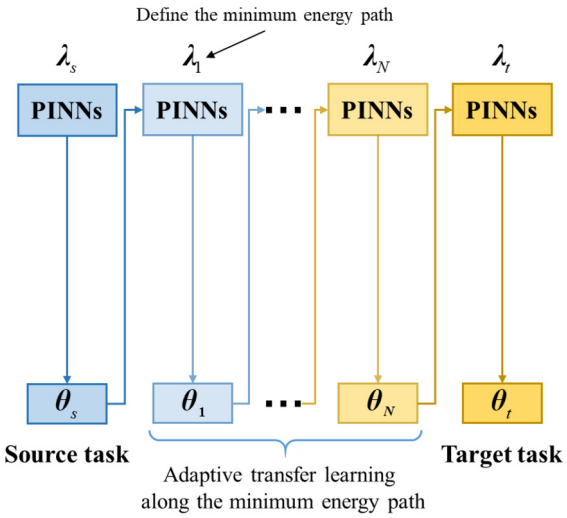


Fig. 3. The basic idea of AtPINN, PINN starts from the source task described by $\boldsymbol{\lambda}_s$, is sequentially trained to the PINN described by $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, ..., \boldsymbol{\lambda}_N$, and finally reaches the target task described by $\boldsymbol{\lambda}_t$.

next PINN described by $\boldsymbol{\lambda}_{i+1}$. Thus, initialized by the source task, PINN is sequentially trained to the PINN described by $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, ..., \boldsymbol{\lambda}_N$, and finally reaches the target PINN.

To implement the process, a new loss function is designed in the new parameter space V for AtPINN according to the equation (9):

$$\begin{aligned}\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) &= w_{\mathcal{F}}\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + w_{\mathcal{I}}\mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + w_{\mathcal{B}}\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + w_{\mathcal{P}}\mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda}) \\ &= \mathcal{L}_{PINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + w_{\mathcal{P}}\mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda})\end{aligned} \tag{10}$$

where $w_{\mathcal{P}}$ is the weight of $\mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda})$ with $w_{\mathcal{P}} = \frac{1}{2}k$, and $\mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda})$ is expressed as:

$$\mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda}) = \|\boldsymbol{\lambda} - \boldsymbol{\lambda}_t\|^2 \tag{11}$$

where $\boldsymbol{\lambda}$ is initialized by $\boldsymbol{\lambda}_s$ and the optimization target is $\boldsymbol{\lambda}_t$, and $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$, $\mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$, and $\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$ can be redefined:

$$\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{1}{N_{\mathcal{F}}}\sum_{i=1}^{N_{\mathcal{F}}} \|\mathcal{F}(\boldsymbol{x}^{(i)}, t^{(i)}; \hat{u}, \boldsymbol{\lambda})\|^2 \tag{12a}$$

$$\mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{1}{N_{\mathcal{I}}}\sum_{i=1}^{N_{\mathcal{I}}} \|\mathcal{I}(\boldsymbol{x}^{(i)}, t^{(i)}, h^{(i)}; \hat{u}, \boldsymbol{\lambda})\|^2 \tag{12b}$$

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{1}{N_{\mathcal{B}}}\sum_{i=1}^{N_{\mathcal{B}}} \|\mathcal{B}(\boldsymbol{x}^{(i)}, t^{(i)}, g^{(i)}; \hat{u}, \boldsymbol{\lambda})\|^2 \tag{12c}$$

$\boldsymbol{\lambda}$ can be updated by the gradient descent procedure:

$$\boldsymbol{\lambda}_{i+1} = \boldsymbol{\lambda}_i - \eta\nabla_{\boldsymbol{\lambda}}\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \tag{13}$$

where η is the learning rate. At the ith epoch, PINN is trained to approximate the solution described by $\boldsymbol{\lambda}_i$. At the next epoch of $i+1$, $\boldsymbol{\lambda}_i$ is updated to $\boldsymbol{\lambda}_{i+1}$ by equation (13), and then PINN is optimized to approximate the next solution described by $\boldsymbol{\lambda}_{i+1}$

According to equations (8) and (10), the loss function of AtPINN is defined in the new parameter space of V and guides the PINN to adaptively transfer from the source to the target task. $\boldsymbol{\lambda}$ is utilized to control the direction of this path, and $\boldsymbol{\theta}$ is free to adapt the change imposed by $\boldsymbol{\lambda}$. Three points need to be noted:

(I) If the loss is required to converge to a global minimum value for each update of $\boldsymbol{\lambda}$, it will consume a lot of computational time and resources. Thus, this manuscript only needs the loss to converge to a low level, which, on the one hand, can ensure the stability of the optimization, and on the other hand, can effectively reduce the computational time.

(II) Not the AtPINN loss $\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda})$, but the PINN loss $\mathcal{L}_{PINN}(\boldsymbol{\theta}, \boldsymbol{\lambda})$ in equation (10) needs to be always kept at the low level in the parameter space of V_1 during the training process.

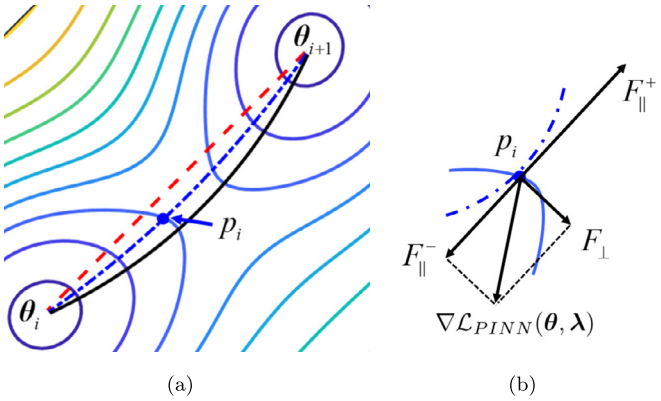


Fig. 4. (a) The minimum energy path on the two dimensional loss surface, the black solid line represents the minimum energy path, the red dashed line represents the linear path, and the blue dash dot line represents a path between the minimum energy path and linear path. (b) The gradient behaviors of different loss terms.

(III) $\boldsymbol{\lambda}$ guides the direction of transfer learning, and the parameter loss $\mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda})$ remains a certain value until the optimization goal is reached.

3.5.1. Determine the weight assignment of each loss term

The loss function of PINN contains multiple terms that compete with each other during the training process. AtPINN has one more loss term than PINN, and in addition, $\boldsymbol{\lambda}$ can exist in different loss terms with various forms, which implies more complicated competition and may lead to an unstable optimization process. For example, when $\boldsymbol{\lambda}$ exists in the PDEs, there are usually a large number of sampling points distributed in the computational domain that vigorously dominate the update of neural networks. When $\boldsymbol{\lambda}$ exists in initial or boundary conditions, only a small amount of points are sampled and powerlessly update neural networks. To ensure AtPINN has good robustness in different applications, it is necessary to assign suitable weight to each loss term.

Now, we determine the weights ($w_{\mathcal{F}}, w_{\mathcal{I}}, w_{\mathcal{B}}$, and $w_{\mathcal{P}}$) by analyzing the gradient behaviors of each loss term during the training process. First, consider the gradient of $\mathcal{L}_{AtPINN}(\boldsymbol{\theta},\boldsymbol{\lambda})$ with respect to $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{AtPINN} = w_{\mathcal{F}}\frac{\partial\mathcal{L}_{\mathcal{F}}}{\partial\boldsymbol{\theta}} + w_{\mathcal{I}}\frac{\partial\mathcal{L}_{\mathcal{I}}}{\partial\boldsymbol{\theta}} + w_{\mathcal{B}}\frac{\partial\mathcal{L}_{\mathcal{B}}}{\partial\boldsymbol{\theta}} \tag{14}$$

Since the parameters $\boldsymbol{\lambda}$ exist only in the PDEs, the gradient of $\mathcal{L}_{AtPINN}(\boldsymbol{\theta},\boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$:

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L}_{AtPINN} = w_{\mathcal{F}}\frac{\partial\mathcal{L}_{\mathcal{F}}}{\partial\boldsymbol{\lambda}} + w_{\mathcal{P}}\frac{\partial\mathcal{L}_{\mathcal{P}}}{\partial\boldsymbol{\lambda}} \tag{15}$$

Suppose $F_{\perp} = \nabla_{\boldsymbol{\theta}}\mathcal{L}_{AtPINN}$, $F_{\shortparallel}^{-} = w_{\mathcal{F}}\frac{\partial\mathcal{L}_{\mathcal{F}}}{\partial\boldsymbol{\lambda}}$, and $F_{\shortparallel}^{+} = w_{\mathcal{P}}\frac{\partial\mathcal{L}_{\mathcal{P}}}{\partial\boldsymbol{\lambda}}$. Fig. 4(a) exhibits the minimum energy path on the two dimensional loss surface and gradient behaviors on a point p_i are visualized in Fig. 4(b). They can be divided into two directions: (1) $F_{\perp}$ is perpendicular to the minima energy path, and (2) $F_{\shortparallel}^{-}$ and $F_{\shortparallel}^{+}$ are parallel to the path. Let us focus on their behaviors:

(I) $F_{\shortparallel}^{+}$ is derived from $\mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda})$ and guides the direction of transfer learning. Thus, the direction of the minimum energy path is determined by the local tangent $F_{\shortparallel}^{+}$ to the path and the sign of $F_{\shortparallel}^{+}$ is defined as positive.
(II) $\boldsymbol{\theta}$ is free to adapt to the change imposed on $\boldsymbol{\lambda}$, but this adaptability is limited due to the properties of neural networks, which increases the PINN loss $\mathcal{L}_{PINN}(\boldsymbol{\theta},\boldsymbol{\lambda})$ at each epoch. In other words, the update of $\boldsymbol{\theta}$ is slower than the update of $\boldsymbol{\lambda}$, causing the large $\mathcal{L}_{PINN}(\boldsymbol{\theta},\boldsymbol{\lambda})$. Thus, $F_{\shortparallel}^{-}$ is generated by $\mathcal{L}_{PINN}(\boldsymbol{\theta},\boldsymbol{\lambda})$ and thus its sign is negative, which holds back the transfer of $\boldsymbol{\lambda}$.
(III) The addition of $F_{\shortparallel}^{+}$ and $F_{\shortparallel}^{-}$ determines the change imposed on $\boldsymbol{\lambda}$. As a result, $F_{\perp}$ drives $\boldsymbol{\theta}$ to be gradually updated from the linear path to the minimum energy path.
(IV) In addition, $\nabla\mathcal{L}_{PINN}(\boldsymbol{\theta},\boldsymbol{\lambda}) = F_{\shortparallel}^{-} + F_{\perp}$ suggests the gradient behavior originating from the PINN loss $\mathcal{L}_{PINN}(\boldsymbol{\theta},\boldsymbol{\lambda})$ that also is presented in Fig. 4(b).

In fact, we can imagine a force analysis of a spring to understand the gradient behaviors. The function of $F_{\shortparallel}^{+}$ can be analogized to a driving force applied to a spring, $F_{\shortparallel}^{-}$ is a restoring force originating from the spring, and $F_{\perp}$ is the result of $F_{\shortparallel}^{+}$ and $F_{\shortparallel}^{-}$ acting on the spring.

Neural networks have a limited capacity to adapt to a change or perturbation imposed on the PDEs' parameters. On the one hand, if $\boldsymbol{\lambda}$ changes too much, $F_{\shortparallel}^{+}$ becomes large, and consequently, $\boldsymbol{\lambda}$ is optimized to the target value at a rapid speed, but $\boldsymbol{\theta}$ is updated too slowly to accommodate this change, resulting in a large $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta},\boldsymbol{\lambda})$. It means that the optimization

process deviates from the expected path. On the other hand, if the change of $\boldsymbol{\lambda}$ is too small, a considerable number of epochs are required to reach the target parameters and the optimization process becomes expensive and time-consuming. Moreover, it should be noted that $\boldsymbol{\lambda}$ needs to be updated appropriately to ensure a low loss of $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$. According to these analyses, the weights $w_{\mathcal{F}}$, $w_{\mathcal{I}}$, $w_{\mathcal{B}}$, and $w_{\mathcal{P}}$ can be determined and the loss function is:

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{|\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})|}{c} \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + \mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda}) \tag{16}$$

where weight coefficients $w_{\mathcal{I}}$ and $w_{\mathcal{B}}$ are set as 1 because $\boldsymbol{\lambda}$ exists only in the PDEs, $w_{\mathcal{P}}$ is also set as 1 to ensure the speed of the transfer learning. $w_{\mathcal{F}}$ is determined by the magnitude of $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$ together with a small constant c (such as 0.001). During the training process, $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$ needs to remain at the same order of magnitude as c, that is utilized to approach the minimum energy path. If $|\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})| > c$, the larger $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$, the larger $w_{\mathcal{F}}$ imposed on it. AtPINN will focuses on reducing $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$ to get close to the expected path. If $|\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})| < c$, a penalty is imposed on $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$, and AtPINN pays more attention to the transfer process of the PDEs' parameters.

Based on the above analysis, the principle of weight assignment can be determined: keep the loss terms with the PDEs' parameters (such as $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})$) at a low level. Starting with intelligent initialization, the other loss terms can also be maintained at a low loss where there is little competition between multiple loss terms. Similarly, if the PDEs' parameters only exist in the initial or boundary conditions, the loss function of AtPINN can also be obtained:

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + \frac{|\mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}, \boldsymbol{\lambda})|}{c} \mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda}) \tag{17a}$$

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}) + \frac{|\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}, \boldsymbol{\lambda})|}{c} \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + \mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda}) \tag{17b}$$

3.5.2. Normalization for multiple PDEs' parameters

The PDEs' parameters $\boldsymbol{\lambda}$ often include multiple parameters $[\lambda^1, \lambda^2, ..., \lambda^M]$. AtPINN is initialized with $\boldsymbol{\lambda}_s = \left[\lambda_s^1, \lambda_s^2, \cdots, \lambda_s^M\right]$ and the optimization goal is $\boldsymbol{\lambda}_t = \left[\lambda_t^1, \lambda_t^2, \cdots, \lambda_t^M\right]$. It takes a considerable long training process to transfer each parameter from the source task to the target task one by one. This manuscript takes the normalization technique to transfer multiple parameters simultaneously to reduce training time as well as to avoid over-complicating this problem. For arbitrary parameter λ^m:

$$\lambda = \frac{\lambda^m - \lambda_s^m}{\lambda_t^m - \lambda_s^m} \tag{18}$$

where λ is the normalized parameter that is initialized with 0 for the source task and set to 1 for the target task, λ_s^i and λ_t^i are the ith parameter of source and target parameter, respectively.

Each parameter is embedded into loss function of AtPINN by:

$$\lambda^m = \lambda \left(\lambda_t^m - \lambda_s^m\right) + \lambda_s^m \tag{19}$$

To implement the adaptive update of λ from 0 to 1, the loss terms $\mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda})$ is written as

$$\mathcal{L}_{\mathcal{P}}(\lambda) = \|\lambda - 1\|^2 \tag{20}$$

and λ can be updated by:

$$\lambda_{i+1} = \lambda_i - \eta \left(\sum_{m=1}^{M} \frac{\partial \mathcal{L}_{PINN}(\boldsymbol{\theta}, \boldsymbol{\lambda})}{\partial \lambda^m} \frac{\partial \lambda^m}{\partial \lambda} + \frac{\partial \mathcal{L}_{\mathcal{P}}(\lambda)}{\partial \lambda} \right) \tag{21}$$

where $\boldsymbol{\lambda} = [\lambda^1, \lambda^2, ..., \lambda^M]$.

The process of normalization introduces multiple Jacobian factors $J(\lambda) = [\lambda_t^1 - \lambda_s^1, \lambda_t^2 - \lambda_s^2, ..., \lambda_t^M - \lambda_s^M]$ that impact the weights in equation (16) and (17) when calculating the gradient. To address this effect, the strategy of using the maximum value of the multiple Jacobian factors $max(J(\lambda))$ is adopted to adjust the weights.

For multiple parameters $[\lambda^1, \lambda^2, ..., \lambda^M]$, equation (16) becomes

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{|\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda})|}{c \cdot max(J(\lambda))} \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + \mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda}) \tag{22}$$

where $J(\lambda) = [\lambda_t^1 - \lambda_s^1, \lambda_t^2 - \lambda_s^2, ..., \lambda_t^M - \lambda_s^M]$, and equation (17) becomes

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + \frac{|\mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}, \boldsymbol{\lambda})|}{c \cdot max(J(\lambda))} \mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda}) \tag{23a}$$

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}) + \frac{|\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}, \boldsymbol{\lambda})|}{c \cdot max(J(\lambda))} \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}, \boldsymbol{\lambda}) + \mathcal{L}_{\mathcal{P}}(\boldsymbol{\lambda}) \tag{23b}$$

3.6. Discussion

This manuscript introduces a novel method AtPINN by leveraging the concept of the MEP. AtPINN implements transfer learning along the expected low-loss path, ensuring the stability of the optimization. During the process, The PDEs' parameters are adaptively updated from the source to the target parameters to guide the optimization of PINN from the source to the target task. It is important to highlight that unlike the MEP, which requires the loss function to reach its minimum value, AtPINN only requires the loss function to reach a relatively small order of magnitude (e.g., 0.01 or 0.001).

4. Results and discussion

In this section, AtPINN is employed to achieve transfer learning cases with a large range of parameter variation for solving a series of complex problems (such as the sharp local gradient and high-frequency problems), and the classical PINN and tPINN are also executed for comparison. All cases are performed by Pytorch 1.2 with a GPU of NVIDIA GeForce RTX 2060. More details are available on GitHub at https://github.com/liuyangair/AtPINN.

4.1. Boundary layer simulation of ordinary differential equation

4.1.1. Problem setup

The first case is to use AtPINN to simulate the boundary layer, which exists widely in computational mechanics. For example, the boundary layer in fluid mechanics is the thin region of flow adjacent to a solid surface due to the influence of viscosity. In addition, the edge layer in solid mechanics, the interface layer between the atmosphere and the ocean or land, etc., are the same phenomenon. In the inner region of the boundary layer, the velocity gradient is very large. However, PINN is not robust in representing this sharp local gradient in a broad computational domain. In this case, based on a smooth initial solution, we try to use AtPINN to implement a transfer learning case with a large range of parameter variation to model boundary layer phenomena.

To simplify the simulation, consider a second-order ordinary differential equation (ODE):

$$\varepsilon y'' + y' + y = 0,\ 0 < x < 1 \tag{24a}$$

$$y(0) = a,\ y(1) = b \tag{24b}$$

where ε is a small parameter ($\varepsilon \ll 1$), a, b is the boundary conditions ($a, b \geq 0$ and $a \neq be$), $y(x)$ is the solution of the ODE. The solution of this ODE appears as a thin boundary layer at $x = 0$ as the parameter tends to 0 ($\varepsilon \to 0$).

The ODE will be solved with the classic PINN, tPINN, and AtPINN in a 1-D computational domain of $x = [0, 1]$ with 500 points uniformly distributed. The boundary conditions are set as $a = 0$ and $b = 1$. PINN is initialized with random weights, and tPINN is initialized with reference weights that have been learned previously. Their loss functions are the same:

$$\mathcal{L}_{PINN}(\boldsymbol{\theta}) = \mathcal{L}_F(\boldsymbol{\theta}) + \mathcal{L}_B(\boldsymbol{\theta}) \tag{25}$$

AtPINN is also initialized with reference weights and its loss function is:

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \varepsilon) = \frac{|\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \varepsilon)|}{c}\mathcal{L}_F(\boldsymbol{\theta}, \varepsilon) + \mathcal{L}_B(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{P}}(\varepsilon) \tag{26a}$$

$$\mathcal{L}_{\mathcal{P}}(\varepsilon) = \|\varepsilon - \varepsilon_t\|^2 \tag{26b}$$

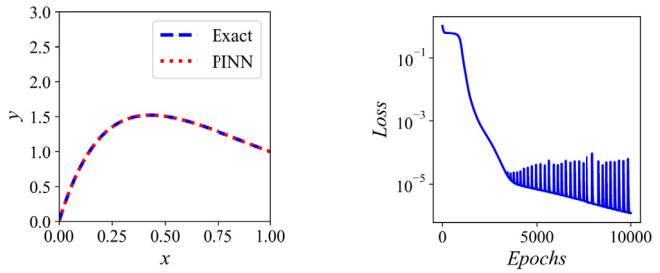
where c is set to be 0.001, ε is initialized by ε_s (source task), and ε_t is the ODE parameter of the target task.

4.1.2. Results for the ODE boundary layer

The results are divided into two parts. The first part is to generate a source PINN. The fully-connected neural networks with 3 hidden layers and 30 neurons per layer are employed. The Adam optimizer with the initial learning rate of 0.0001 is employed and the maximum optimization epoch is set as 10,000. PINN is used to solve the ODE with $\varepsilon = 0.2$. Fig. 5 plots the results. The solution is smooth and gentle in the whole computational domain, and the boundary layer has not yet appeared.

To demonstrate the advantages of AtPINN, the second part is to use PINN, tPINN, and AtPINN to solve the ODE with $\varepsilon = 0.02$ and 0.002, respectively. The network structure of all cases is the same as the source PINN. PINN is set with the same optimizer as the source PINN. The LBFGS algorithm runs with low time memory requirement but provides faster convergence than first-order methods. It is worth mentioning that the LBFGS algorithm is based on Newton's method, hence the computational accuracy strongly depends on the initial guess. Transfer learning strategy enhances the convergence of training based on a smart initialization, so tPINN and AtPINN are set with the LBFGS optimizer. In addition, the results of the PINN optimized by the LBFGS algorithm with random initialization are also presented for comparison. Note that the optimization algorithm for all the transfer learning cases is the LBFGS algorithm to minimize the loss function.

Fig. 6 gives the comparison of all cases. First, for the results of PINN, both Adam and LGFBS optimizers achieve the same results, which indicates that the results are independent of the optimization algorithm. Good predictions are given



(a) Comparison (b) Loss

Fig. 5. Comparison of the exact solution and PINN solution with $\varepsilon = 0.2$ (source task).

when $\varepsilon = 0.02$, corresponding to a boundary layer thickness of 0.08. However, PINN fails to solve the case with $\varepsilon = 0.002$, because the boundary layer (the thickness of 0.01) becomes much thinner than the case with $\varepsilon = 0.02$. Second, tPINN shows good results in the case of $\varepsilon = 0.02$. However, tPINN is not effective in the case of $\varepsilon = 0.002$, which may be due to that the distance between $\varepsilon_{\text{target}}$ and $\varepsilon_{\text{source}}$ is too large and the target task is too different from the source task. Finally, the proposed method, AtPINN not only exhibits good results in the case of $\varepsilon = 0.02$ but also in the case of $\varepsilon = 0.02$, which proves the ability and potential for simulating the sharp local gradient problems.

4.1.3. Analysis

A. Comparison of losses for different PINN cases Fig. 7 plots the loss of the PINN with different parameters and optimizers. For the Adam optimizer, PINN solves the case of $\varepsilon = 0.2$ with fast convergence (no more than 2000 epochs). In the case of $\varepsilon = 0.02$, although good results are obtained, the optimization processing becomes difficult with the same settings (more than 8000 epochs). In the case of $\varepsilon = 0.002$, the loss does not decrease and remains on the order of 10e-1, which indicates that PINN cannot address this case. For the LGFBS optimizer, PINN solves the case of $\varepsilon = 0.02$ with less than 1000 epochs, and it can not deal with the case of $\varepsilon = 0.002$ either.

B. Comparison of losses between PINN and tPINN Fig. 8 shows the comparison of losses between PINN and tPINN when $\varepsilon = 0.02$. Two differences deserve special attention. First, tPINN has a high loss at the beginning of training, which is caused by the large range of the sudden variation in the parameter. A more detailed explanation of this phenomenon will be discussed through the visualization of tPINN. Second, tPINN exhibits fast convergence, twice as fast as the PINN, indicating the advantage of transfer learning techniques.

C. Comparison of three tPINN cases with different ranges of parameter variation To further explain the large loss at the beginning of the $\varepsilon = 0.02$ case, another tPINN case with $\varepsilon = 0.15$, which is much closer to the source task, is implemented. Fig. 9 plots the results, and it can be seen that a high-precision solution is obtained. Now, we will compare three tPINN cases with different ranges of parameter variation. Fig. 10 gives the loss comparison of tPINN cases with $\varepsilon = 0.15$, 0.02, and 0.002. At the beginning of training, the losses of $\varepsilon = 0.02$ case and $\varepsilon = 0.002$ case are comparable, and they are both higher than the loss of $\varepsilon = 0.15$ case. As ε decreases, the range of parameter variation from the source task becomes large and the transfer process from the source to the target task becomes difficult. The loss of $\varepsilon = 0.15$ case converges to the order of 10e-4 within 200 epochs, and the loss of $\varepsilon = 0.02$ case converges within 500 epochs. However, since tPINN can not address the case of $\varepsilon = 0.002$, the corresponding loss stays above 10e-1.

D. Visualization of optimization process for tPINN The differences among three tPINN cases with different ranges of parameter variation will be further discussed by visualization analysis. Fig. 11 gives the two-dimensional and three-dimensional visualization, exhibiting the solution of some epochs in the optimization process of tPINN. tPINN is initialized by the source task which is presented as a blue solid line in Fig. 11(a) and 11(b). In the tPINN case of $\varepsilon = 0.15$ showed in Fig. 11(a) and 11(c), the solution features of the source task are preserved because of the small variation between the source and target parameter. After a few epochs, the optimization direction is determined so that the solution can be quickly transferred to the target task. In the tPINN case with $\varepsilon = 0.02$ presented in Fig. 11(b) and 11(d), the tPINN solution starts with the source task, and drops suddenly to a non-physical solution, which can explain the reason why tPINN has a large loss at the beginning. The features of the source task are not fully utilized and only the feature representing the local gradient near $x = 0$ is retained. First, tPINN takes a long time of training to find the optimization direction that lies in the sharp gradient near $x = 0$. Then, tPINN gradually optimizes the weights, preferentially satisfying the local gradients near $x = 0$, until the target task is reached. Through the visualization analysis, it can be seen that the advantage of transfer learning is not fully exploited in the tPINN case with a large range of parameter variation. In some sense, tPINN is equivalent to the re-optimization process, which does not apply to many problems.

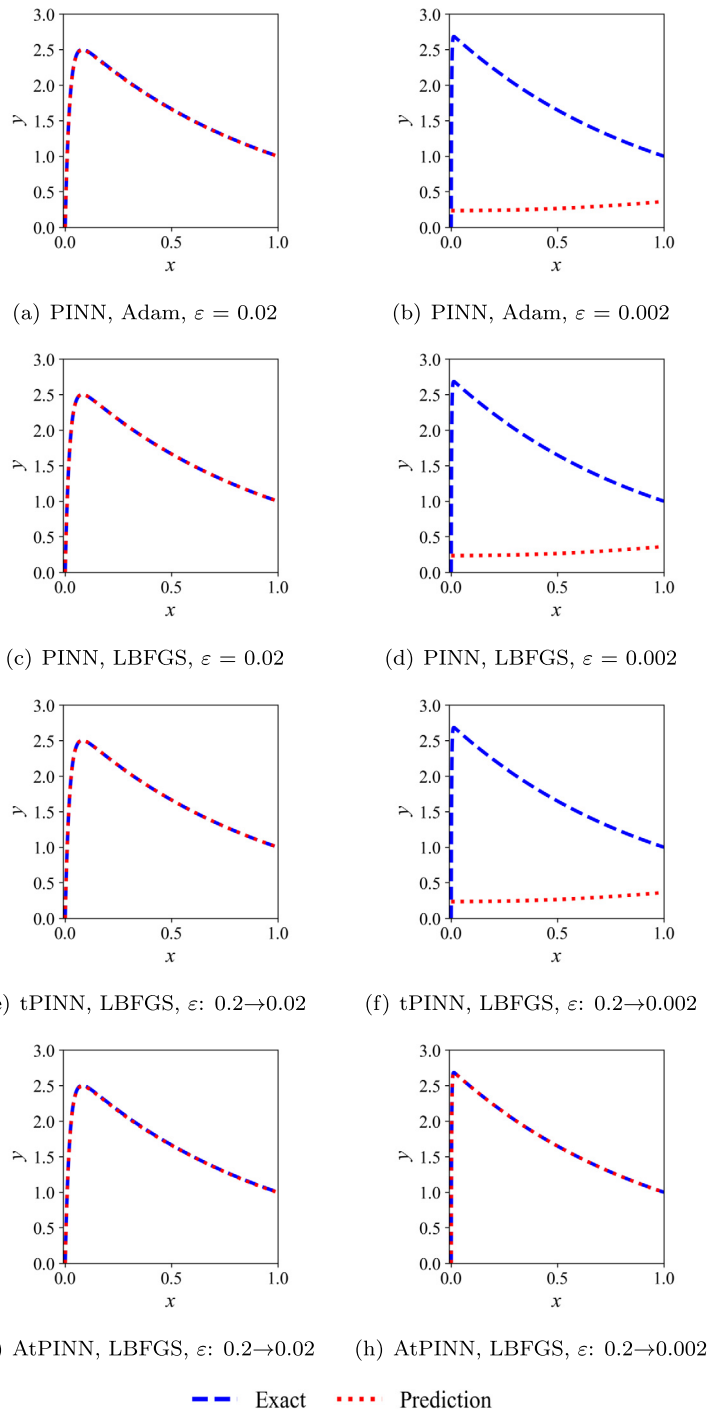
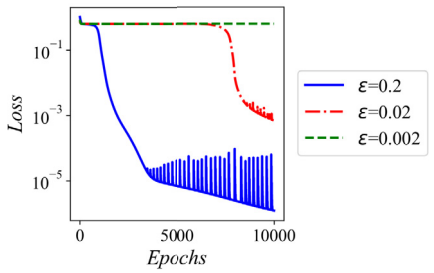


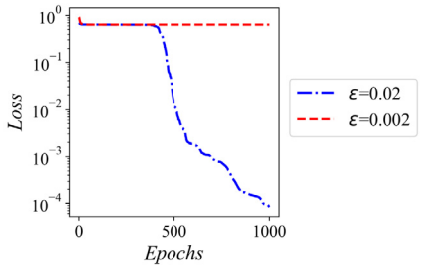
Fig. 6. Results of different methods for the ODE boundary layer (blue dashed line represents the exact solution, and red dotted line represents the solution of the neural networks).

E. Comparison of various losses for AtPINN To describe the optimization process of AtPINN in detail, three various losses are defined: $\mathcal{L}_{AtPINN} = \mathcal{L}_{\mathcal{F}} + \mathcal{L}_{\mathcal{B}} + \mathcal{L}_{\mathcal{P}}$, $\mathcal{L}_{AtPINN_{ODE}} = \mathcal{L}_{\mathcal{F}} + \mathcal{L}_{\mathcal{B}}$, and $\mathcal{L}_{AtPINN_{\varepsilon}} = \mathcal{L}_{\mathcal{P}}$. In addition, the loss of tPINN defined as $\mathcal{L}_{tPINN}$ is also used for comparison. Fig. 12 gives the results. We first analyze the variation of ε and various losses of AtPINN, which can be divided into four stages (also see in Table 1):

(I) The first stage is the few epochs at the beginning of training, where ε decreases rapidly from the initial value. Since neural networks are initialized by the source PINN and the weights satisfy the equations with the initial parameter,



(a) Adam optimizer



(b) LBFGS optimizer

Fig. 7. Loss comparison of the PINN with different parameters and optimizers.

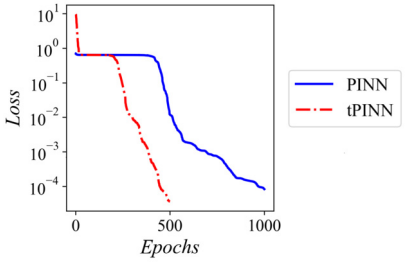


Fig. 8. Comparison of losses between PINN and tPINN when $\varepsilon = 0.02$ (LBFGS optimizer).

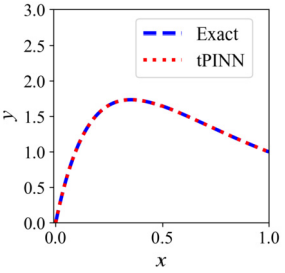


Fig. 9. Comparison of exact and tPINN solution (LGFBS, ε: 0.2→0.15).

$\mathcal{L}_{AtPINN_{ODE}}$ is very small in the order of 10e-6. The loss is mainly contributed by $\mathcal{L}_{AtPINN_{\varepsilon}}$ at this time, which is used to update ε, resulting in the rapid decrease of ε.

(II) In the second stage, the update of ε leads to an update of the neural network weights, which is also fed back to $\mathcal{L}_{AtPINN_{\varepsilon}}$ and thus ε returns to the position near the initial value. In this stage, the optimization algorithm balances the loss terms $\mathcal{L}_{AtPINN_{ODE}}$ and $\mathcal{L}_{AtPINN_{\varepsilon}}$, which can be considered as the preparing for parameter adaptive learning.

(III) In the third stage, ε is updated by the $\mathcal{L}_{AtPINN_{\varepsilon}}$ term from ε_s to ε_t, correspondingly, neural networks are trained from the source task to the target task. As seen in Fig. 12(c) and 12(d), the loss is mainly contributed by $\mathcal{L}_{AtPINN_{\varepsilon}}$ that is an order of magnitude higher than $\mathcal{L}_{AtPINN_{ODE}}$. More importantly, $\mathcal{L}_{AtPINN_{ODE}}$ is in the order of 10e-2 to 10e-3, which means that the solution at each epoch during the AtPINN optimization process is close to the exact solution of the

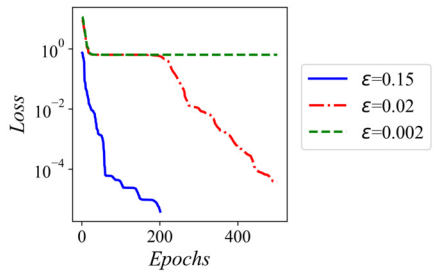
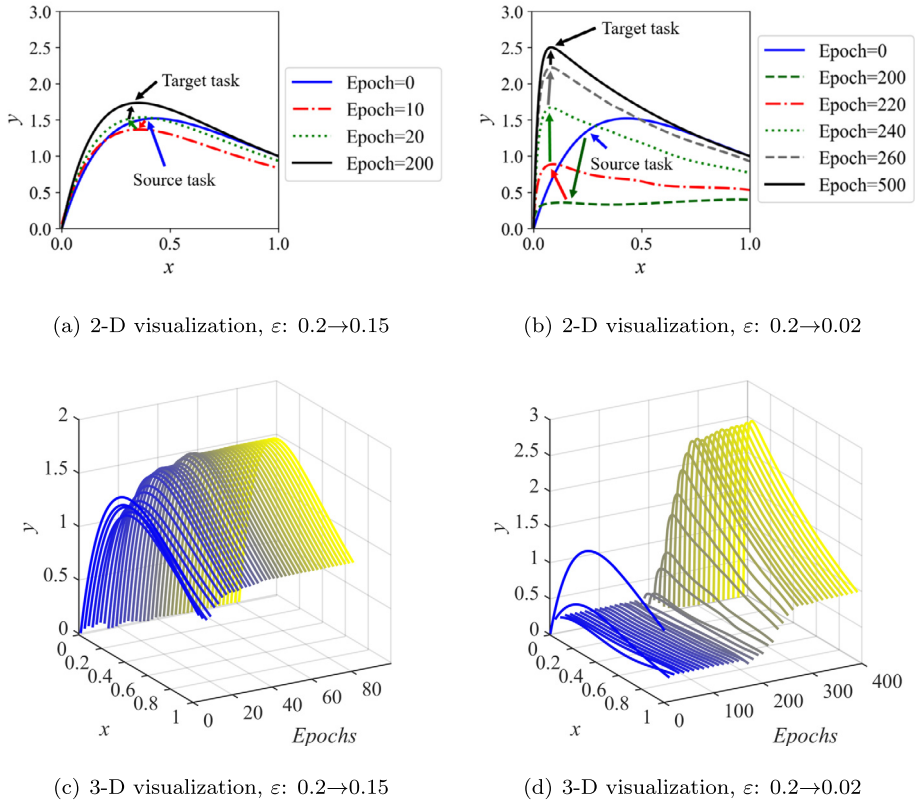


Fig. 10. Comparison of three tPINN case with different range of parameter variation.



(a) 2-D visualization, ε: 0.2→0.15

(b) 2-D visualization, ε: 0.2→0.02

(c) 3-D visualization, ε: 0.2→0.15

(d) 3-D visualization, ε: 0.2→0.02

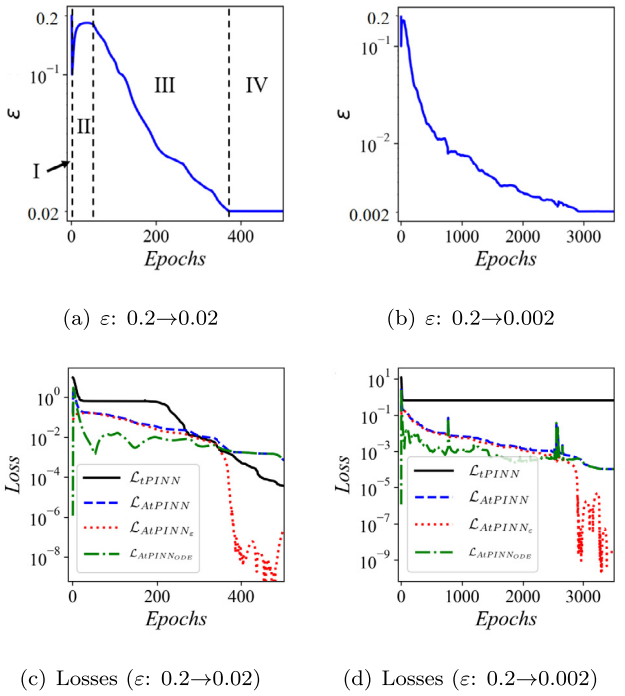
Fig. 11. 2-D and 3-D visualization during the optimization process of tPINN.

corresponding parameter. Fig. 13 presents the solution at epoch = 100 with the corresponding parameter of 0.11965. At this epoch, the solution is close to the current exact solution, demonstrating the above analysis.

(IV) The fourth stage is the end of optimization. ε reaches the target parameter and $\mathcal{L}_{AtPINN_\varepsilon}$ rapidly reduces to near 0. Contrary to the third stage, the loss mainly comes from the $\mathcal{L}_{AtPINN_{ODE}}$. The purpose of optimization in this stage is to obtain a high-precision solution.

Second, we compare the differences between tPINN and AtPINN. Although $\mathcal{L}_{tPINN}$ and $\mathcal{L}_{AtPINN_{ODE}}$ have the same loss definition, $\mathcal{L}_{tPINN}$ is two orders of magnitude larger than $\mathcal{L}_{AtPINN_{ODE}}$ (see the second and third stages in Fig. 12(c) and 12(d)). The tPINN falls the local optimal solution (see Fig. 11(d) and Fig. 6(f)) and therefore leads to a high loss. For AtPINN, $\mathcal{L}_{AtPINN_{ODE}}$ is the loss defined in section 3.5.1 that needs to be always kept at the low level, and the optimization process is performed along the designed low-loss path. Hence, the AtPINN solution at each epoch is very close to the physical solution, which will be illustrated in the next analysis.

F. Visualization of optimization process for AtPINN The 2-D and 3-D visualization of the optimization process for AtPINN is presented in Fig. 14. Unlike tPINN, AtPINN maintains the characteristics of the source task solution. Based on a smooth initial solution, AtPINN is gradually optimized from the source task to the target task. Besides, the solution at each epoch during the AtPINN optimization process conforms to the basic form of the physical solution. In the 3-D visualization, the



(a) ε: 0.2→0.02 (b) ε: 0.2→0.002

(c) Losses (ε: 0.2→0.02) (d) Losses (ε: 0.2→0.002)

Fig. 12. The variation of ε and the comparison of various losses during the AtPINN optimization process.

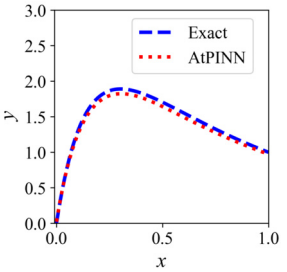


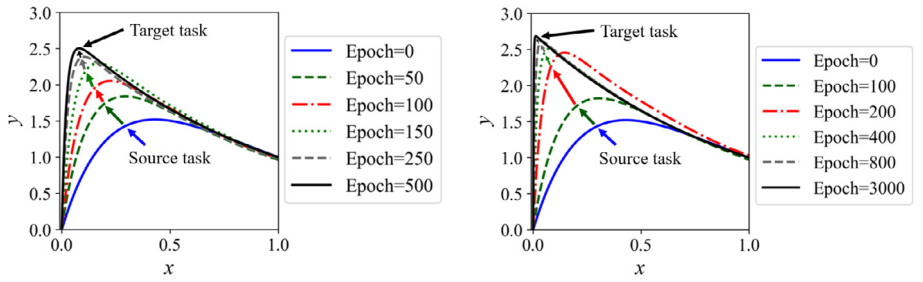
Fig. 13. Comparison of exact and AtPINN solution with the epoch = 100 (the corresponding $\varepsilon = 0.11965$).

Table 1
Four Stages during the AtPINN optimization process.

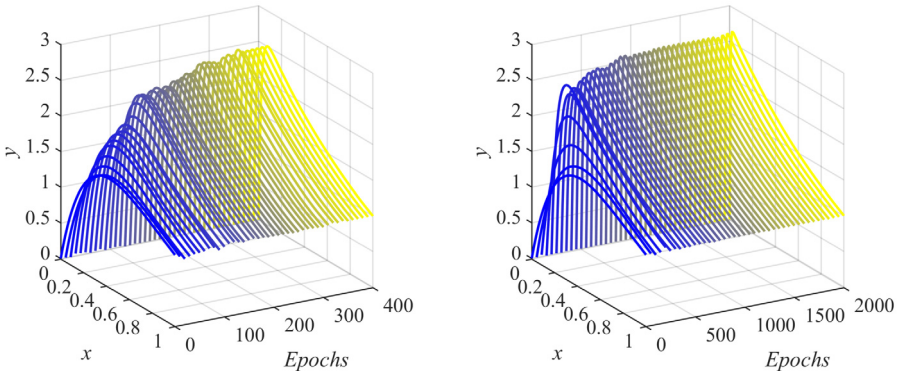
Stage	Epoch	Evaluation
I	At the beginning	Start training AtPINN Loss is mainly contributed by $\mathcal{L}_{AtPINN_\varepsilon}$
II	A few epochs after the start	Balance $\mathcal{L}_{AtPINN_{ODE}}$ and $\mathcal{L}_{AtPINN_\varepsilon}$ They have the same order of magnitude
III	Main training	Parameter adaptive learning $\mathcal{L}_{AtPINN_\varepsilon}$ is much larger than $\mathcal{L}_{AtPINN_{ODE}}$
IV	At the end	Leading to an accurate solution Loss is mainly contributed by $\mathcal{L}_{AtPINN_{ODE}}$

optimization can be considered to proceed along a smooth surface, which fully demonstrates the advantages of AtPINN. As shown in Fig. 14(b) and 14(d), AtPINN perfectly solves the task of $\varepsilon = 0.002$ that tPINN cannot solve, transferring the solution from $\varepsilon = 0.2$ to $\varepsilon = 0.002$.

G. The visualization of MEP for AtPINN The adaptive transfer process is implemented along a designed or expected low-loss path, which can ensure the stability of the training process. To demonstrate this process in AtPINN, the MEP is visualized through the following steps: (1) Extract the weights of each step in the optimization process of AtPINN $[\boldsymbol{\Theta}_0, \boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_{N+1}]$. (2) Apply principal component analysis (PCA) based on the literature [40] to select the two most explanatory directions $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ (basis vectors for PCA). (3) Calculate the coordinates of the starting point $\boldsymbol{\Theta}_0$ and ending point $\boldsymbol{\Theta}_{N+1}$ of the MEP

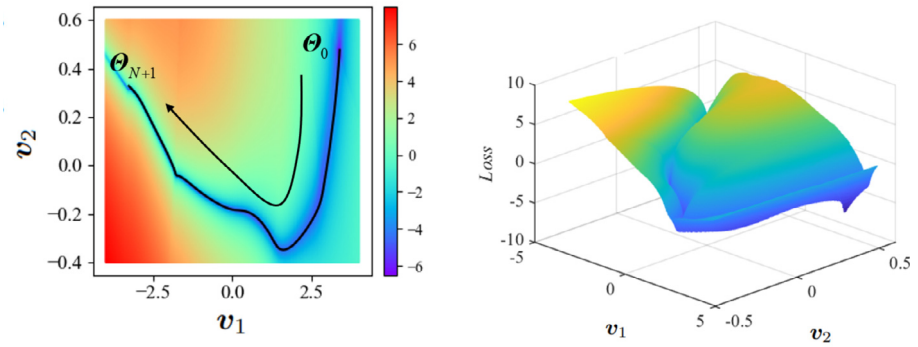


(a) 2-D visualization, ε: 0.2→0.02 (b) 2-D visualization, ε: 0.2→0.002



(c) 3-D visualization, ε: 0.2→0.02 (d) 3-D visualization, ε: 0.2→0.002

Fig. 14. 2-D and 3-D visualization during the optimization process of AtPINN.

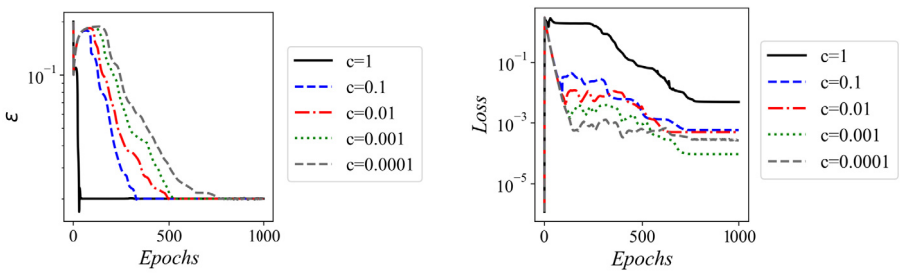


(a) 2-D visualization (b) 3-D visualization

Fig. 15. 2-D and 3-D visualization of MEP for AtPINN.

based on $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, which are (3.39, 0.48) and (-3.28, 0.33), respectively. (4) Select a reference point $\boldsymbol{\Theta}^r$ with coordinates (0, 0). (5) Plot a loss function of the form $\mathcal{L}_{AtPINN}(\boldsymbol{\Theta}^r + \alpha \boldsymbol{v}_1 + \beta \boldsymbol{v}_2)$ for $\alpha \in [-4, 4]$ and $\beta \in [-0.4, 0.6]$, as shown in Fig. 15. It can be observed that along the MEP, the loss function is in a valley with low loss, while in other areas, the loss function has larger values, which is consistent with Fig. 2.

H. The influence of c The selection of c can have a significant impact on the process of adaptive transfer learning. Here we aim to investigate and analyze the effects of this choice in detail. To this end, different values of c, namely 1, 0.1, 0.01, 0.001, and 0.0001, are considered, and AtPINN is employed to solve the ODE case with $\varepsilon = 0.02$ based on the source task ($\varepsilon = 0.2$). Fig. 16 presents the variation of ε and $\mathcal{L}_{AtPINN_{ODE}}$ during the optimization process of AtPINN for different values of c. When c is set to 1, the weight $w_{\mathcal{F}}$ is small and the optimization mainly focuses on updating ε at the beginning. This enables ε to rapidly reach the target value. However, due to slow updates of $\boldsymbol{\theta}$ that cannot adapt to such quick changes of ε, $\mathcal{L}_{AtPINN_{ODE}}$ remains at the order of 1. These findings suggest that the optimization process of AtPINN has deviated from the



(a) ε: 0.2→0.02 (b) $\mathcal{L}_{AtPINN_{ODE}}$

Fig. 16. The variation of ε and $\mathcal{L}_{AtPINN_{ODE}}$ during the optimization process of AtPINN for different values of c.

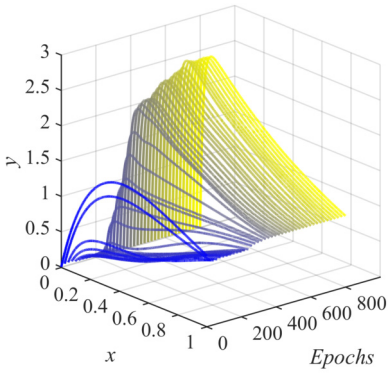


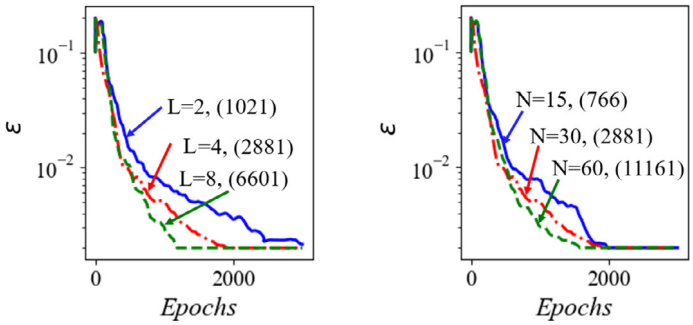
Fig. 17. 3-D visualization during the optimization process of AtPINN ($c = 1$).

expected path, as demonstrated by a three-dimensional visualization (shown in Fig. 17). AtPINN can carry out the adaptive transfer learning from the source task to the target task for c values ranging from 0.1 to 0.0001. As the value of c decreases, the order of magnitude of $\mathcal{L}_{AtPINN_{ODE}}$ will gradually decrease, which in turn requires more epochs for the transfer process. This suggests that the transfer process can be controlled by adjusting the weight of loss terms containing parameters. If the value of c is too large, AtPINN may deviate from the expected path, while if c is too small, the transfer process may become exceedingly slow. To ensure a stable and efficient transfer process, this manuscript recommends setting c to either 0.01 or 0.001.

I. Performance of AtPINN with different network configurations We investigate the adaptive transfer performance of AtPINN across different scales of neural networks. Specifically, consider five neural networks with parameter counts of 1021, 766, 2881, 11161, and 6601, respectively, each with different numbers of hidden layers and nodes. The number of hidden layers ranges from 2 to 8, and the number of nodes per hidden layer ranges from 15 to 60. These networks can be divided into two categories: (1) the first category has a fixed number of hidden layers (4 layers) but varying numbers of nodes (15, 30, and 60); (2) the second category has a fixed number of nodes per layer (30 nodes) but varying numbers of hidden layers (2, 4, and 8). For the source task, these networks are trained to solve the ODE case with $\varepsilon = 0.2$ using the Adam optimizer with an initial learning rate of 0.0001 and a maximum epoch of 10,000. For the target task, the networks are trained to solve the case with $\varepsilon = 0.002$ using the L-BFGS optimizer with an initial learning rate of 0.1, a maximum epoch of 3000, and a c set to 0.01. Besides, PINN and tPINN with the same optimizer are also used to solve the same target task. Table 2 gives the average absolute error of PINN, tPINN, and AtPINN for the case with $\varepsilon = 0.002$. For all five neural networks, AtPINN achieves high accuracy in solving the ODE case with $\varepsilon = 0.2$, whereas PINN and tPINN failed to solve the problem. These results demonstrate the excellent performance of AtPINN across different network sizes. Fig. 18 gives adaptive transfer process of ε for different network configurations. Besides, the adaptive transfer process of AtPINN exhibits faster convergence when the number of neurons in the hidden layer increases, given that the number of hidden layers is fixed. Similarly, when the number of neurons in the hidden layers is held constant, the adaptive transfer process of AtPINN displays quicker convergence as the number of hidden layers increases. Moreover, the acceleration effect of increasing the network depth is superior to that of increasing the number of nodes, which is in line with the conventional understanding that deeper neural networks have greater expressive power. These findings suggest that increasing the number of trainable parameters in neural networks can accelerate the adaptive transfer process of AtPINN.

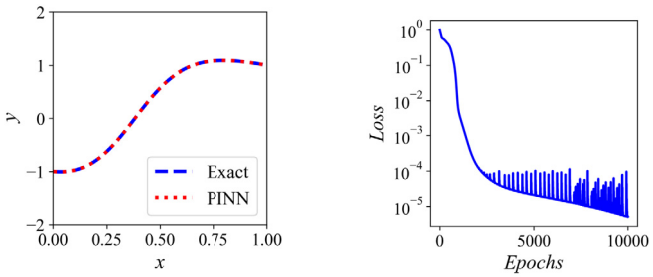
Table 2
The average absolute error of PINN, tPINN, and AtPINN for the case with $\varepsilon = 0.002$.

Hyperparameters	PINN	tPINN	AtPINN
Layers=2, Nodes=30	1.4689	1.4688	0.0017
Layers=4, Nodes=15	1.4688	1.4687	0.0020
Layers=4, Nodes=30	1.4688	1.4688	0.0009
Layers=4, Nodes=60	1.4688	1.4688	0.0027
Layers=8, Nodes=30	1.4688	1.4688	0.0028



(a) N=30 (b) L=4

Fig. 18. Adaptive transfer process of ε for different network configurations.



(a) Comparison (b) Loss

Fig. 19. Comparison of the exact solution and PINN solution with $\varepsilon = 0.2$ (source task, shock layer).

4.2. Shock layer simulation of ordinary differential equation

4.2.1. Problem setup

In this section, AtPINN is exploited to deal with another problem in fluid mechanics, the shock wave problem. Consider such an equation:

$$\varepsilon y'' + yy' + e^{1-x}y = 0,\ 0 < x < 1 \tag{27a}$$

$$y(0) = a,\ y(1) = b \tag{27b}$$

where ε is a small parameter ($\varepsilon \ll 1$).

In the boundary condition set with $a = -1, b = 1$, the solution of equation (27) exhibits interior layer behavior, which is also used to simulate the shock wave in aerodynamics. The computational domain and the sampling data are similar to section 4.1.1. The loss functions of PINN and tPINN are defined as equation (25), and AtPINN as equation (26a) and (26b).

4.2.2. Results for the ODE shock layer

First, the source PINN with $\varepsilon = 0.2$ is generated. The hyperparameters of the neural networks and settings of the optimizer are the same as those in Section 4.1.1. Fig. 19 gives the results. The solution varies gently at the boundary and slowly increases from -1 to 1 throughout the interval. Due to the large value of ε, the solution presents a smooth curve, and the shock wave layer is not obvious. The PINN solution is very close to the exact solution, and the loss is reduced to below 10e-5, which illustrates the effectiveness of PINN under the current parameter.

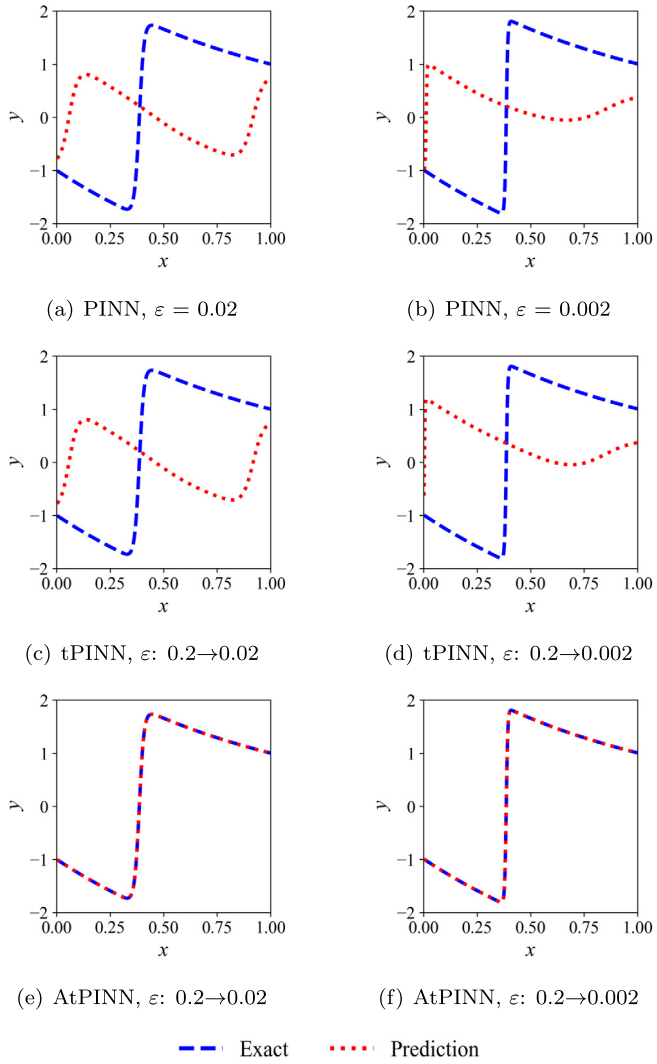


Fig. 20. Results of different methods for the ODE shock layer (blue dashed line represents the exact solution, and red dotted line represents the solution of neural networks).

Then, the three methods are used to solve the ODE with ε = 0.02 and 0.002, respectively. All the cases are optimized by the LBFGS algorithm. Fig. 20 gives the comparison results. The results of PINN fall into local optimal solutions and cannot accurately simulate the shock layer. Due to the large range of parameter variation, tPINN shows the same results as PINN. Only AtPINN achieves satisfactory results, which indicates the ability to capture the shock layer.

Fig. 21 gives the variation of ε and the comparison of four losses. The variations of ε and various losses are the same as the analysis in section 4.1.3. Note that the losses of two tPINN cases are larger than 10e-1, because the corresponding solutions converge to non-physical solutions. Fig. 22 gives the visualization analysis of the AtPINN training process. During the optimization process, the smooth and gentle solution is gradually optimized to the solution with sharp inner layer under the guidance of the parameter, and finally reaches the target solution.

4.3. Boundary layer simulation of partial differential equation

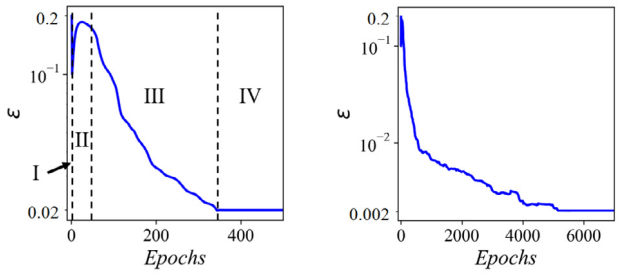
4.3.1. Problem setup

In this case, a parabolic equation problem with the boundary layer will be illustrated. Consider the problem of solving:

$$\varepsilon \frac{\partial u}{\partial t} - \varepsilon^2 \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x} - 2t = 0 \tag{28a}$$

$$u(x, 0) = 0 \tag{28b}$$

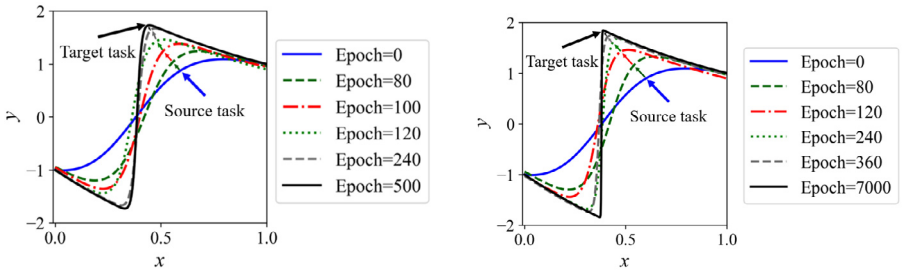
$$u(0, t) = u(1, t) = 0 \tag{28c}$$



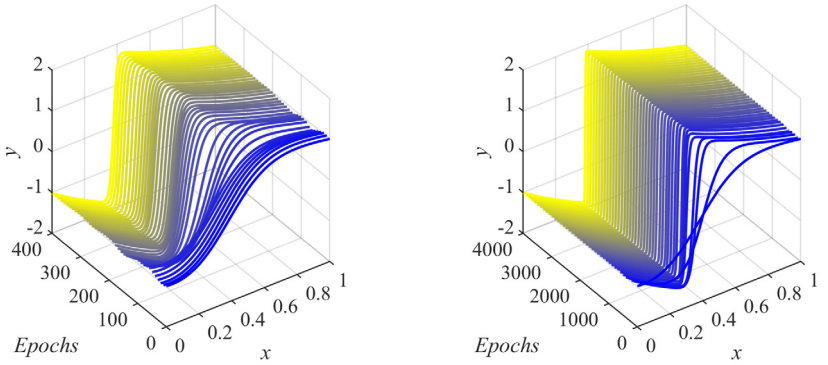
(a) ε: 0.2→0.02 (b) ε: 0.2→0.002

(c) Losses (ε: 0.2→0.02) (d) Losses (ε: 0.2→0.02)

Fig. 21. The variation of ε and the comparison of various losses during the AtPINN optimization process (shock layer).

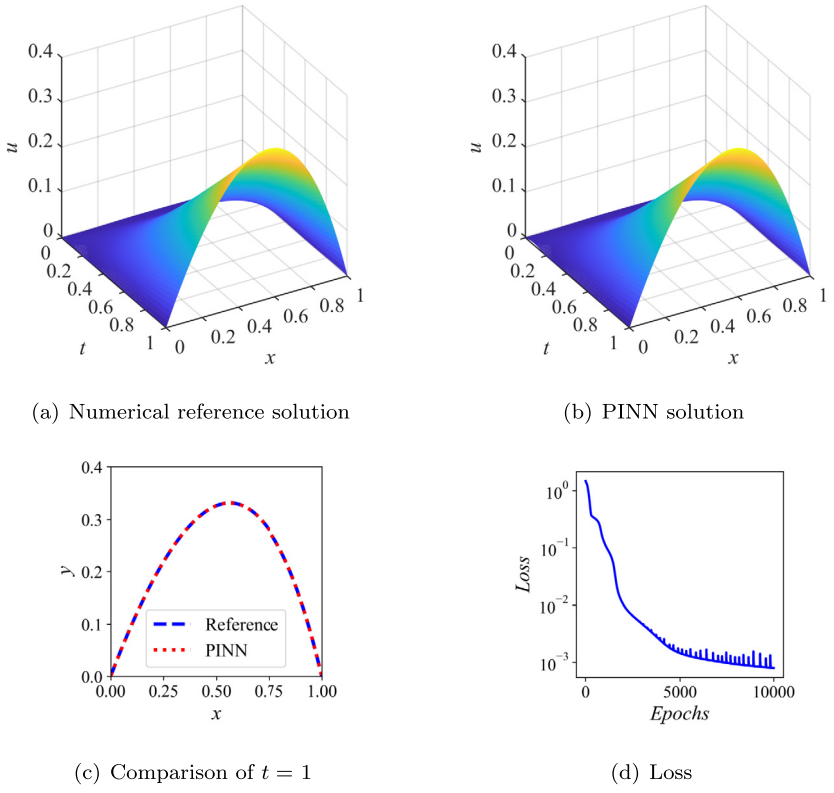


(a) 2-D visualization, ε: 0.2→0.02 (b) 2-D visualization, ε: 0.2→0.002



(c) 3-D visualization, ε: 0.2→0.02 (d) 3-D visualization, ε: 0.2→0.002

Fig. 22. 2-D and 3-D visualization during the optimization process of AtPINN (shock layer).



(a) Numerical reference solution (b) PINN solution

(c) Comparison of $t = 1$ (d) Loss

Fig. 23. Comparison of the numerical reference solution and PINN solution with $\varepsilon = 0.8$.

where ε is a small parameter ($\varepsilon \ll 1$), $u(x,t)$ is the solution to this equation, and the computational domain is $0 < x < 1,\ 0 < t \leq 1$. $u(x,t)$ has a boundary layer at $x = 1$ and the gradient within the boundary layer becomes large as t increases.

The loss function of PINN and tPINN is defined as:

$$\mathcal{L}_{PINN}(\boldsymbol{\theta}) = \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) \tag{29}$$

and the loss function of AtPINN is:

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \varepsilon) = \frac{|\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \varepsilon)|}{c}\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \varepsilon) + \mathcal{L}_{\mathcal{I}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{P}}(\varepsilon) \tag{30a}$$

$$\mathcal{L}_{\mathcal{P}}(\varepsilon) = \|\varepsilon - \varepsilon_t\|^2 \tag{30b}$$

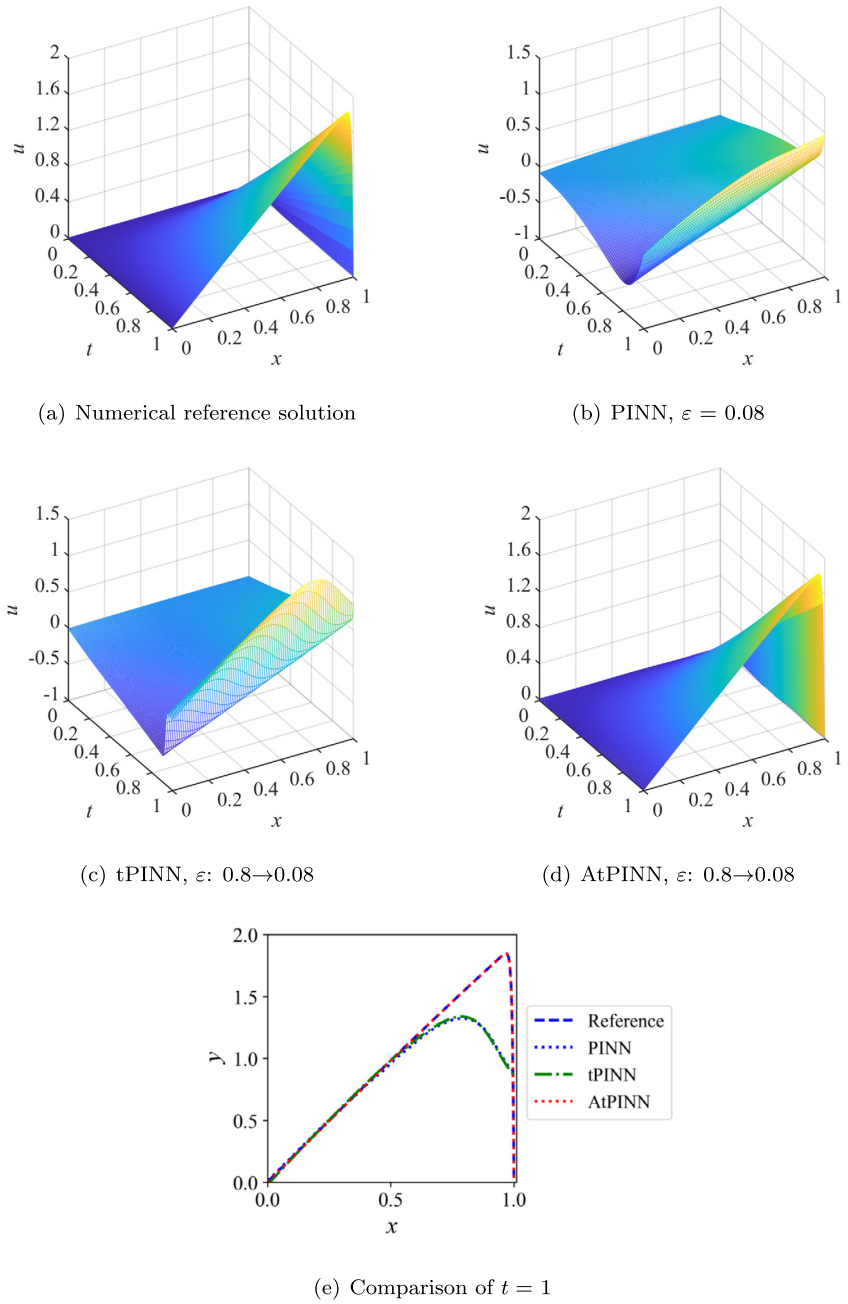
where c is set as 0.001.

4.3.2. Results for the PDE boundary layer

Firstly, the source PINN with $\varepsilon = 0.8$ is prepared. Fully-connected neural networks with 4 hidden layers and 50 neurons per layer are employed. The Adam optimizer with the same settings in section 4.1 is employed. The numerical reference solution and the PINN solution the are given in Fig. 23(a) and 23(b), respectively. The solution appears as a smooth surface in the entire computational domain, and the shape of their solutions is similar. Fig. 23(c) shows the comparison at $t = 1$, where the solution of PINN is very close to the reference solution. The loss in Fig. 23(c) reduces to below 10e-3. These illustrate the effectiveness of PINN in solving the case with ε = 0.8.

Then, PINN, tPINN, and AtPINN are utilized to solve the equation with ε = 0.08. The optimizers for the three cases are LBFGS algorithm. The results and numerical reference solution are plotted in Fig. 24. Besides, the solutions of $t = 1$ for all cases are compared in Fig. 24(e). As we can see, the solution of ε = 0.08 appears as a boundary layer at $x = 1$, and the gradient in the boundary layer becomes progressively larger as time increases. The PINN and tPINN cases fall into the local optimal solution, which is far from the numerical solution. Only AtPINN achieves good result, which is quite identical to numerical solution in Fig. 24(e).

In this case, define $\mathcal{L}_{AtPINN} = \mathcal{L}_{\mathcal{F}} + \mathcal{L}_{\mathcal{I}} + \mathcal{L}_{\mathcal{B}} + \mathcal{L}_{\varepsilon}$, $\mathcal{L}_{AtPINN_{PDE}} = \mathcal{L}_{\mathcal{F}} + \mathcal{L}_{\mathcal{I}} + \mathcal{L}_{\mathcal{B}}$, and $\mathcal{L}_{AtPINN_{\varepsilon}} = \mathcal{L}_{\mathcal{P}}$. Together with $\mathcal{L}_{tPINN}$, the variation of the parameter and various losses are presented in Fig. 25. The variations can be divided into four stages as analyzed in section 4.1.3. The difference is that AtPINN takes a long time to lead to a high accuracy solution in the fourth stage, in that the two dimensional solution requires a more refined optimization. Since tPINN fails to solve the current



(a) Numerical reference solution

(b) PINN, $\varepsilon = 0.08$

(c) tPINN, ε: 0.8→0.08

(d) AtPINN, ε: 0.8→0.08

(e) Comparison of $t = 1$

Fig. 24. Results of different methods for the PDE boundary layer.

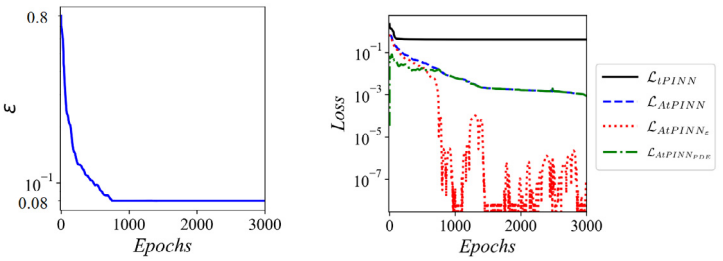
problem, the loss $\mathcal{L}_{tPINN}$ is consistently not reduced. The visualization of the solution of $t = 1$ during the optimization is given in Fig. 26. It can be seen that the solution, the two dimensional surface, is also transferred gradually from the source task to the target task in a smooth form.

4.4. Boundary layer simulation of steady incompressible Navier-Stokes equations

4.4.1. Problem setup

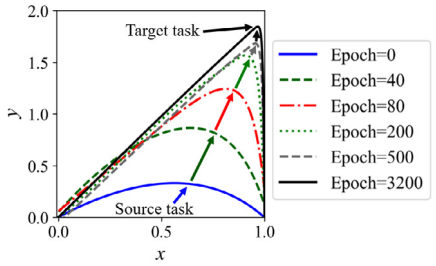
In this case, AtPINN is implemented to model boundary layer phenomena when flowing through the NACA0012 airfoil. Consider the steady incompressible Navier-Stokes equations:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{31a}$$

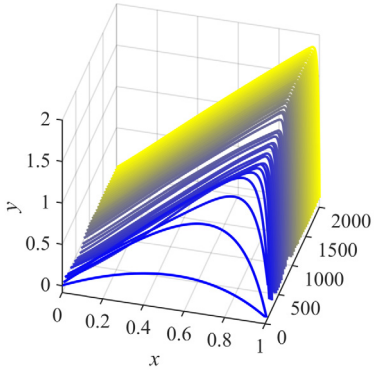


(a) ε: 0.8→0.08 (b) Losses (ε: 0.8→0.08)

Fig. 25. The variation of ε and the comparison of various losses during the AtPINN optimization process.



(a) 2-D visualization, ε: 0.8→0.08



(b) 3-D visualization, ε: 0.8→0.08

Fig. 26. 2-D and 3-D visualization during the optimization process of AtPINN (t = 1).

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\frac{\mu}{\rho}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{31b}$$

$$u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial y}+\frac{\mu}{\rho}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \tag{31c}$$

where ρ is the density, p is the pressure, $[u; v]$ are the velocity components, and μ is dynamic viscosity.

As shown in Fig. 27, the length l of the airfoil is 1 m, and the computational domain is $-0.5 < x < 1.5$ and $-0.5 < y < 0.5$. The input data consists of 12000 points in the flow field, 240 points on the airfoil, and 600 points on rectangular boundaries ($x = -0.5$, $x = 1.5$, and $y = \pm 0.5$). The pressure data of the airfoil is provided for solving this case. Besides, the pressure and velocity information of the rectangular boundaries are also provided. When the Reynolds number ($Re = \frac{\rho U l}{\mu}$, $U = \sqrt{u^2 + v^2}$) becomes large, the thin boundary layer occurs at the region around the airfoil, where a large number of points are sampled.

The loss function of PINN and tPINN is:

$$\mathcal{L}_{PINN}(\boldsymbol{\theta}) = \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) \tag{32}$$

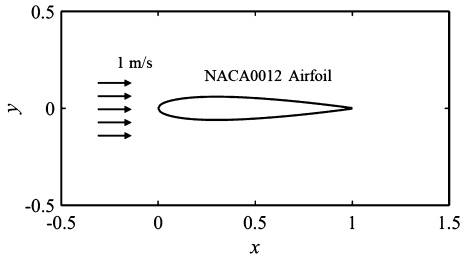
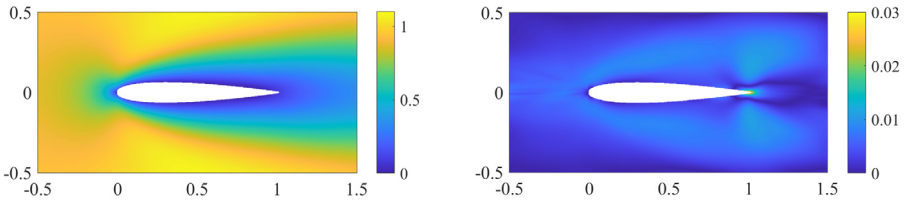


Fig. 27. The computational domain of flowing through NACA0012 airfoil.



(a) The velocity magnitude (b) The absolute error

Fig. 28. The velocity magnitude and absolute error for PINN under $Re = 50$, $\mu = 0.02$ (unit: m/s).

where the $\mathcal{L}_{\mathcal{D}}$ represents the loss term of observation data and its weight is set as 1. And the loss function of AtPINN is:

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \mu) = \frac{|\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \mu)|}{c}\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}, \varepsilon) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{P}}(\mu) \tag{33a}$$

$$\mathcal{L}_{\mathcal{P}}(\mu) = \|\mu - \mu_t\|^2 \tag{33b}$$

where c is set as 0.001, μ is used to indirectly represent the Reynolds number, and its initialization is μ_s.

4.4.2. Results

Neural networks of 4 layers with 100 nodes are employed and the LBFGS optimizer is chosen. The freestream velocity is 1 m/s with the angle of attack of 0°, and the density is 1 kg/m^3. First, the source PINN with $Re = 50$, $\mu = 0.02$ is prepared. Fig. 28 gives the velocity magnitude $U = \sqrt{u^2 + v^2}$ contour and its absolute error (refer to the CFD solution). The variation of velocity magnitude is gentle in the entire flow field. The source task shows an accurate result and the maximum error is no more than 0.03 m/s which is three percent of the freestream velocity.

Then, the source PINN is leveraged as initialization for solving the case with $Re = 1000$, $\mu = 0.001$ through tPINN and AtPINN. When Re increases to 1000, a thin boundary layer appears at the region near the airfoil. Fig. 29 visually compares the velocity magnitudes and errors contour. Both two methods basically exhibit the accurate flowfield of flowing through the airfoil. However, when focusing on the boundary layer region around the airfoil, AtPINN achieves a smaller error than tPINN in simulating the sharp local gradient (see in Fig. 29(b) and Fig. 29(d)). The maximum absolute error of tPINN exceeds 0.03 m/s, and the error of AtPINN is within 0.02 m/s. Define $\mathcal{L}_{AtPINN} = \mathcal{L}_{\mathcal{F}} + \mathcal{L}_{\mathcal{B}} + \mathcal{L}_{\mathcal{D}} + \mathcal{L}_{\mu}$, $\mathcal{L}_{AtPINN_{PDE}} = \mathcal{L}_{\mathcal{F}} + \mathcal{L}_{\mathcal{B}}$, and $\mathcal{L}_{AtPINN_{\mu}} = \mathcal{L}_{\mathcal{P}}$ in this case. Fig. 30 presents the variation of the three losses and $\mathcal{L}_{tPINN}$. $\mathcal{L}_{AtPINN}$ shows an evidently higher loss at the beginning, because μ is suddenly changed but the weights $\boldsymbol{\theta}$ have hot been updated. $\mathcal{L}_{AtPINN_{PDE}}$ is less than 0.01 throughout the whole training process and is always lower than $\mathcal{L}_{tPINN}$. These further demonstrate the advantage of AtPINN in solving problems with the sharp gradient in broad computational domains.

4.5. Laplace equation with high frequency

4.5.1. Problem setup

In this case, AtPINN is employed to directly solve the Laplace equation to model the transfer learning case under multiple parameters. In the spatial domain $0 < x < 1,\ 0 < t \leq 1$, the Laplace equation is considered:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \ \ \forall (x, y) \in \mathcal{U} \tag{34a}$$

$$u = g(x, y) \ \ \forall (x, y) \in \partial\mathcal{U} \tag{34b}$$

where $g(x, y) = sin(-\omega_1 \pi y)e^{-\omega_2 \pi x}$, $\omega_i \in \mathbb{R}$ that corresponds to the eigenfunctions of the Laplace equation to keep the arithmetic simple. High frequencies are expected to be challenging to learn compared to lower frequencies, and thus we set

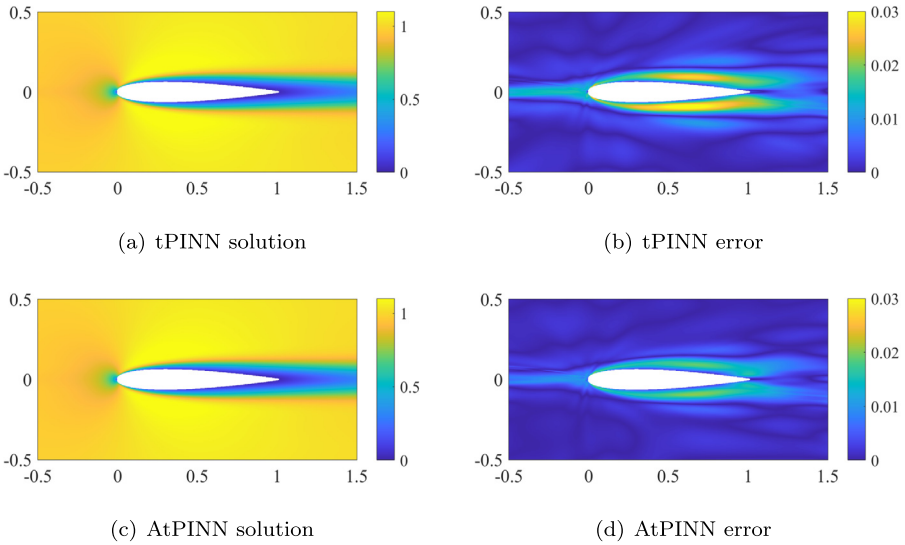


Fig. 29. Comparison of the tPINN and AtPINN results with $Re = 1000$, $\mu = 0.001$ (unit: m/s).

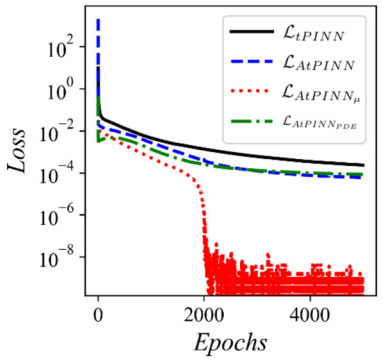


Fig. 30. The comparison of various losses for the case with $Re = 1000$, $\mu = 0.001$.

$\omega_1 = 6, \omega_2 = 3$ to lead to complex solutions. Besides, two parameters ω_1, ω_2 exist in the boundary condition, which also results in the complexity of solving this problem.

Another difference is that AtPINN is leveraged to directly address the above problem without intelligent initialization. Random initialization as the direct training is executed, which is different from the setup of transfer learning. In this set, AtPINN is expected to first learn the low-frequency solution and then be gradually guided by two parameters to obtain the high-frequency solution. Thus ω_1 and ω_2 are set to 0 as the initial value, and the target values are 6 and 3, respectively. According to section 3.5.2, ω_1 and ω_2 can be transformed to the interval of [0, 1] using normalization:

$$\omega = \frac{\omega_1 - 0}{6 - 0} = \frac{\omega_2 - 0}{3 - 0} \tag{35}$$

The loss function of AtPINN is:

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}, \omega) = \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + \frac{\mathcal{L}_{\mathcal{B}}}{c}\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}, \omega) + \mathcal{L}_{\mathcal{P}}(\omega) \tag{36a}$$

$$\mathcal{L}_{\mathcal{P}}(\omega) = \|\omega - 1\|^2 \tag{36b}$$

where c is set as 0.00017, and ω is initialized to 0.

As a comparison, PINN is also used to solve this problem, and the loss function is:

$$\mathcal{L}_{AtPINN}(\boldsymbol{\theta}) = \mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) \tag{37}$$

4.5.2. Results

Fully-connected neural networks with 6 hidden layers and 75 neurons per layer are employed. 22500 points are randomly sampled in the computational domain and 1200 points are uniformly distributed on the boundary. The LBFGS optimizer with

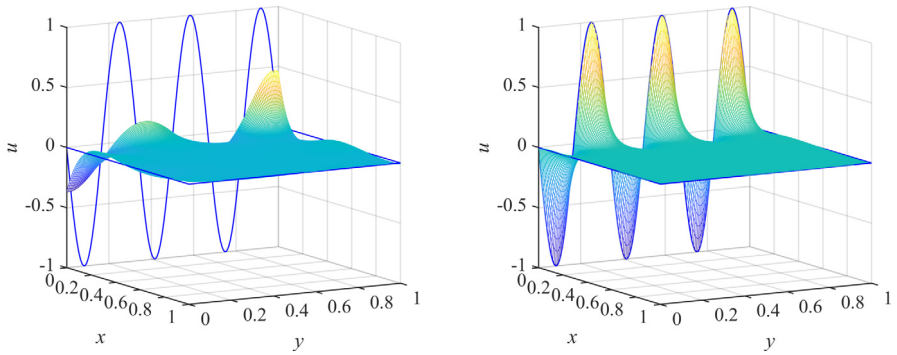


Fig. 31. Comparison of the PINN solution and AtPINN solution with $\omega_1 = 6, \omega_2 = 3$.

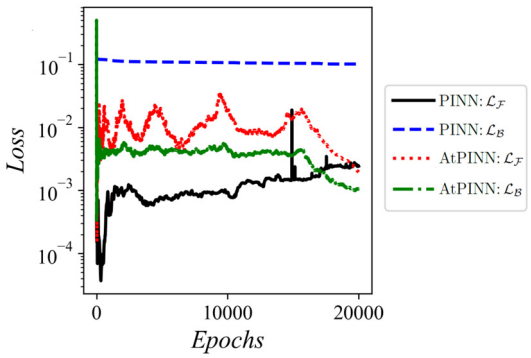


Fig. 32. The PDE and boundary losses of the two methods.

a learning rate of 0.5 is employed and the maximum iteration is 20000. Fig. 31 illustrates this difference in accuracy for the case of $\omega_1 = 6\pi$ and $\omega_2 = 3\pi$. The traditional PINN shows a large error that is several orders of magnitude larger than the theoretical optimum. Obviously, the PINN solution falls into the local optimal solution and is not consistent with the physical law. With the same settings, the AtPINN result is consistent with the boundary condition accurately, as one would expect to exhibit a low error.

Fig. 31(a) presents various losses in different methods. For PINN, the PDE loss is an order of magnitude larger than the loss of the boundary condition, illustrates that PINN focuses excessively on satisfying the PDE rather than the boundary condition. For AtPINN, the losses of PDE and the boundary condition are basically in the same order of magnitude (see Fig. 32). Define $\mathcal{L}_{AtPINN} = \mathcal{L}_{\mathcal{F}} + \mathcal{L}_{\mathcal{B}} + \mathcal{L}_{\omega}$, $\mathcal{L}_{AtPINN_{PDE}} = \mathcal{L}_{\mathcal{F}} + \mathcal{L}_{\mathcal{B}}$, and $\mathcal{L}_{AtPINN_{\omega}} = \mathcal{L}_{\mathcal{P}}$. Together with $\mathcal{L}_{PINN}$, these loss terms are displayed in Fig. 33. With the random initialization, $\mathcal{L}_{AtPINN_{PDE}}$ is at the order of 1 that dividing a small constant c, causing a high loss of $\mathcal{L}_{AtPINN}$. At the same time, neural networks are trained to find a solution to satisfy the Laplace equation. After a few epochs, $\mathcal{L}_{AtPINN_{PDE}}$ is at a low level of 0.01 meaning that the physical solutions are obtained. Since then, $\mathcal{L}_{AtPINN_{PDE}}$ is much smaller than $\mathcal{L}_{PINN}$ and the training process of AtPINN is carried out along the designed path to reach the final model. Moreover, as is exhibited in Fig. 34, AtPINN first learns the solution of $\omega_1 = 1, \omega_2 = 0.5$ at the epoch of 160 without the intelligent initialization, and then learns the complex solutions as ω_1 and ω_2 increase. This means that the adaptive transfer learning can be a particular optimization strategy where AtPINN learns the simple solution at the beginning of the training process and then is gradually transferred to solve the complex task. These further demonstrate the potential of AtPINN and extend the application scenarios of PINN.

5. Conclusions

This manuscript proposes an AtPINN based on the concept of the minimum energy path. The transfer learning process is implemented along the designed or expected low-loss path, ensuring the training process's stability. AtPINN is employed to achieve transfer learning cases with a large range of parameter variation to simulate five complex problems. The results show that AtPINN shows promising potential to solve these problems and effectively extend the application of PINN. Here, we summarize the main properties of this new transfer learning framework as follows:

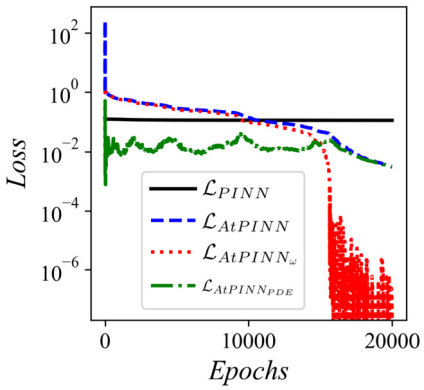


Fig. 33. The PINN loss and various losses of AtPINN.

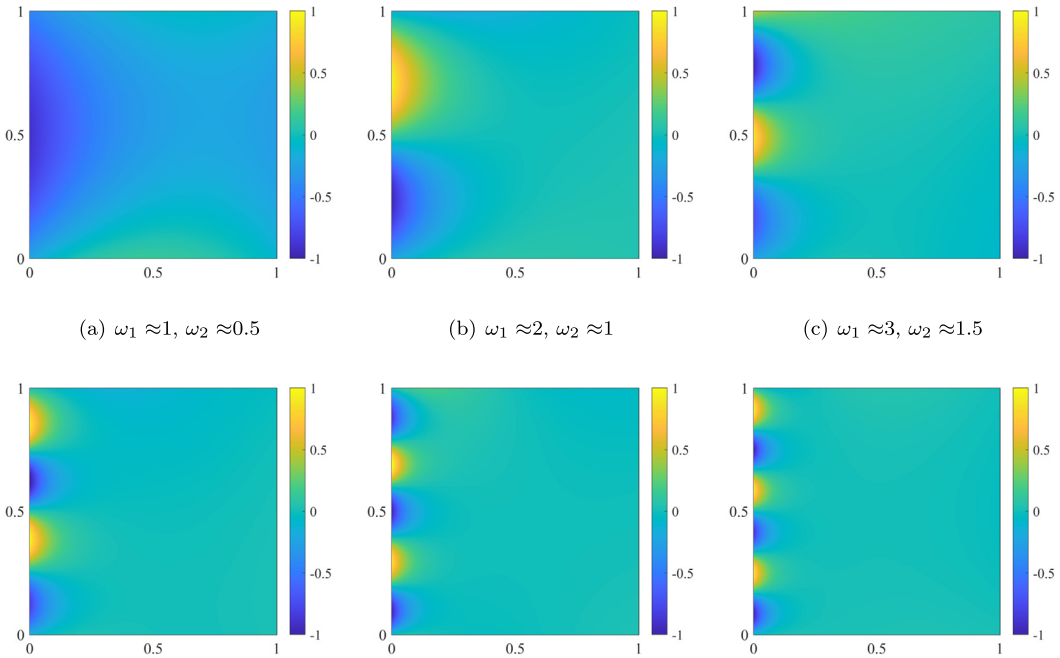


Fig. 34. The solutions during the optimization process of AtPINN. (a): epoch=160, (b): epoch=600, (c): epoch=1400, (d): epoch=3000, (e): epoch=9000, and (f): epoch=18000.

(I) The PDEs' parameters are treated as the optimizable variables, together with the neural network weights forming a new weight set. We develop a feasible minimum energy path for PINN in the new parameter space by fine-tuning the PDEs' parameters.

(II) The minimum energy path for PINN is utilized to redesign the loss function of PINN and develop an adaptive transfer learning method. During the transfer learning process, the PDEs' parameters are updated adaptively from the source to the target parameter, which can guide the transfer of PINN from the source task to the target task. Different from the previous transfer learning studies focusing on knowledge transfer at the feature level, AtPINN emphasizes the fact that the loss is always kept at a low level near the minima, which can effectively ensure the stability of the training process.

(III) The gradient behaviors of different loss terms are analyzed to determine the principle of weight assignment: keep the loss terms with the PDEs' parameters at a low level. Besides, the normalization is leveraged to extend AtPINN to accommodate the transfer learning cases with multiple parameters.

(IV) AtPINN can achieve transfer learning cases with a large range of parameter variation to simulate five complex problems: the boundary layer of ODE, the shock layer of ODE, the boundary layer of PDE, the steady incompressible flowing through NACA0012 airfoil, and the Laplace equation with high frequency. Through the visualization, the training pro-

cess of AtPINN is divided into four stages and the AtPINN solution is always close to the physical solution during the adaptive transfer learning process.

(V) Three transfer learning cases with different range of parameter variations are analyzed through visualization: (1) When the range is small, the source task is strongly correlated with the target task, and the transfer learning can lead to a fast and accurate convergence. (2) When the range becomes large, the source task and the target task are different, and only a few features of the source task are retained. The transfer learning task is equivalent to a retraining process in some sense, but the speed of residual convergence is improved compared with direct learning. (3) When the range is too large and the source task is far from the target task, transfer learning is no longer applicable in such cases.

(VI) In section 4.5, AtPINN can be used to directly solve the Laplace equation with high frequency without intelligent initialization and obtain accurate results. This means that adaptive transfer learning can be regarded as a particular optimization method for PINN where AtPINN learns the simple tasks at the beginning of the training process and then gradually transfer the complex task.

Transfer learning techniques are expected to deal with more complex computational problems, and the proposed method is dedicated to transfer learning with a large range of parameter variation along the optimization path with low loss. The application scenarios can further be expanded, such as parameterized geometry. Besides, it can also be combined with other techniques, such as adaptive sampling, domain decomposition, etc, to solve more complex computational problems. In the future, we will explore the implementations of the proposed ideas on these problems. Furthermore, it is also of interest to explore how this idea can be applied to traditional transfer learning, such as the field of CV and NLP.

CRediT authorship contribution statement

Yang Liu: Conceptualization, Data curation, Investigation, Methodology, Software, Writing – original draft. **Wen Liu:** Data curation, Writing – original draft. **Xunshi Yan:** Investigation, Supervision, Visualization. **Shuaiqi Guo:** Investigation, Supervision, Validation. **Chen-an Zhang:** Funding acquisition, Investigation, Validation.

Declaration of competing interest

All authors disclosed no relevant relationships.

Data availability

Data will be made available on request.

Acknowledgements

The work is supported by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDA17030100), and also supported by the National Science and Technology Major Project of China (ZX069).

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90.
[2] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, IEEE Comput. Intell. Mag. 13 (3) (2018) 55–75.
[3] J.N. Kutz, Deep learning in fluid dynamics, J. Fluid Mech. 814 (2017) 1–4.
[4] A. Froio, R. Bonifetto, S. Carli, A. Quartararo, L. Savoldi, R. Zanino, Design and optimization of artificial neural networks for the modelling of superconducting magnets operation in tokamak fusion reactors, J. Comput. Phys. 321 (2016) 476–491.
[5] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Eng. 361 (2020) 112732.
[6] W. E, W. Ren, E. Vanden-Eijnden, String method for the study of rare events, Phys. Rev. B 66 (5) (2002), https://doi.org/10.1103/PhysRevB.66.052301.
[7] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[8] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2017) 5595–5637.
[9] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026+.
[10] A.D. Jagtap, Z.P. Mao, N. Adams, G.E. Karniadakis, Physics-informed neural networks for inverse problems in supersonic flows, J. Comput. Phys. 466 (2022) 111402, https://doi.org/10.1016/j.jcp.2022.111402.
[11] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218+.
[12] A. Bihlo, R.O. Popovych, Physics-informed neural networks for the shallow-water equations on the sphere, J. Comput. Phys. 456 (2022) 111024.
[13] L. Yuan, Y.Q. Ni, X.Y. Deng, S. Hao, A-pinn: auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations, J. Comput. Phys. 462 (2022) 111260.
[14] E. Taghizadeh, H.M. Byrne, B.D. Wood, Explicit physics-informed neural networks for nonlinear closure: the case of transport in tissues, J. Comput. Phys. 449 (2022) 110781.
[15] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.

[16] V. Dwivedi, B. Srinivasan, Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118.
[17] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (xpinns): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (5) (2020) 2002–2041.
[18] V. Dwivedi, B. Srinivasan, Physics informed extreme learning machine (pielm)-a rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118.
[19] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.
[20] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[21] Y. Guan, A. Chattopadhyay, A. Subel, P. Hassanzadeh, Stable a posteriori les of 2d turbulence using convolutional neural networks: backscattering analysis and generalization to higher re via transfer learning, J. Comput. Phys. 458 (2022) 111090.
[22] X. Jin, S. Cai, H. Li, G.E. Karniadakis, Nsfnets (Navier-Stokes flow nets): physics-informed neural networks for the incompressible Navier-Stokes equations, J. Comput. Phys. 426 (2021) 109951.
[23] Q. Lou, X. Meng, G.E. Karniadakis, Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-BGK formulation, J. Comput. Phys. 447 (2021) 110676.
[24] S.A. Niaki, E. Haghighat, T. Campbell, A. Poursartip, R. Vaziri, Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture, Comput. Methods Appl. Mech. Eng. 384 (2021) 113959.
[25] S. Goswami, C. Anitescu, S. Chakraborty, T. Rabczuk, Transfer learning enhanced physics informed neural network for phase-field modeling of fracture, Theor. Appl. Fract. Mech. 106 (2020) 102447.
[26] E. Haghighat, M. Raissi, A. Moure, H. Gomez, R. Juanes, A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics, Comput. Methods Appl. Mech. Eng. 379 (2021) 113741.
[27] S. Chakraborty, Transfer learning based multi-fidelity physics informed deep neural network, J. Comput. Phys. 426 (2021) 109942.
[28] R. van der Meer, C.W. Oosterlee, A. Borovykh, Optimally weighted loss functions for solving pdes with neural networks, J. Comput. Appl. Math. 405 (2022) 113887.
[29] R. Sun, D. Li, S. Liang, T. Ding, R. Srikant, The global landscape of neural networks: an overview, IEEE Signal Process. Mag. 37 (5) (2020) 95–108.
[30] S. Fort, S. Jastrzebski, Large scale structure of neural network loss landscapes, arXiv:1906.04724 [abs].
[31] F. Draxler, K. Veschgini, M. Salmhofer, F. Hamprecht, Essentially no barriers in neural network energy landscape, in: Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 1309–1318.
[32] T. Garipov, P. Izmailov, D. Podoprikhin, D. Vetrov, A.G. Wilson, Loss surfaces, mode connectivity, and fast ensembling of dnns, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018, pp. 8803–8812.
[33] L. Sagun, U. Evci, V.U. Guney, Y. Dauphin, L. Bottou, Empirical analysis of the hessian of over-parametrized neural networks, arXiv:1706.04454 [abs].
[34] B. Liu, Understanding the loss landscape of one-hidden-layer relu networks, Knowl.-Based Syst. 220 (2021) 106923.
[35] X. Wan, An adaptive high-order minimum action method, J. Comput. Phys. 230 (24) (2011) 8669–8682, https://doi.org/10.1016/j.jcp.2011.08.006.
[36] X. Zhou, W. Ren, W. E, Adaptive minimum action method for the study of rare events, J. Chem. Phys. 128 (10) (2008) 104111, https://doi.org/10.1063/1.2830717, https://www.ncbi.nlm.nih.gov/pubmed/18345881.
[37] D.P. Kingma, J. Ba Adam, A method for stochastic optimization, CoRR, arXiv:1412.6980 [abs].
[38] S. Ruder, An overview of gradient descent optimization algorithms, arXiv:1609.04747 [abs].
[39] R. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization, SIAM J. Sci. Comput. 16 (5) (1995) 1190–1208.
[40] H. Li, Z. Xu, G. Taylor, T. Goldstein, Visualizing the loss landscape of neural nets, in: Neural Information Processing Systems, 2018, pp. 6391–6401.
[41] D. Sheppard, R. Terrell, G. Henkelman, Optimization methods for finding minimum energy paths, J. Chem. Phys. 128 (13) (2008), https://doi.org/10.1063/1.2841941.