

# DEEP NEURAL NETWORKS COMBINING MULTI-TASK LEARNING FOR SOLVING DELAY INTEGRO-DIFFERENTIAL EQUATIONS

WANG Chen-yao, SHI Feng

(*School of Science, Harbin Institute of Technology, Shenzhen 518055, China*)

**Abstract:** Deep neural networks (DNNs) are effective in solving both forward and inverse problems for nonlinear partial differential equations (PDEs). However, conventional DNNs are not effective in handling problems such as delay differential equations (DDEs) and delay integro-differential equations (DIDEs) with constant delays, primarily due to their low regularity at delay-induced breaking points. In this paper, a DNN method that combines multi-task learning (MTL) which is proposed to solve both the forward and inverse problems of DIDEs. The core idea of this approach is to divide the original equation into multiple tasks based on the delay, using auxiliary outputs to represent the integral terms, followed by the use of MTL to seamlessly incorporate the properties at the breaking points into the loss function. Furthermore, given the increased training difficulty associated with multiple tasks and outputs, we employ a sequential training scheme to reduce training complexity and provide reference solutions for subsequent tasks. This approach significantly enhances the approximation accuracy of solving DIDEs with DNNs, as demonstrated by comparisons with traditional DNN methods. We validate the effectiveness of this method through several numerical experiments, test various parameter sharing structures in MTL and compare the testing results of these structures. Finally, this method is implemented to solve the inverse problem of nonlinear DIDE and the results show that the unknown parameters of DIDE can be discovered with sparse or noisy data.

**Keywords:** Delay integro-differential equation; Multi-task learning; parameter sharing structure; deep neural network; sequential training scheme

**2010 MR Subject Classification:** 45J05; 45K05

**Document code:** A

**Article ID:** 0255-7797(2025)01-0013-26

## 1 Introduction

In recent years, machine learning, particularly deep learning, has achieved notable developments in fields such as computer vision, natural language processing, and pattern recognition. Simultaneously, the robust approximate capabilities of deep neural networks (DNNs)

---

\* **Received date:** 2024-08-21

**Accepted date:** 2024-09-02

**Biography:** Wang Chenyao (2000–), male, born at Shijiazhuang Hebei, postgraduate, major in the computational and applied mathematics. E-mail: 22S058003@stu.hit.edu.cn.

**Corresponding author:** Shi Feng.

and their proficiency in handling nonlinear systems have increasingly facilitated their application in computational sciences for solving classical applied mathematical problems, including partial differential equations (PDEs). The physics-informed neural networks (PINNs), as referenced in [1], effectively integrates physical information from PDEs with neural networks, and some libraries have been developed for this method [2, 3]. Successful applications of PINNs have been made which involve PDEs and various engineering problems, such as elastic mechanics [4], fluid mechanics [5, 6], stochastic differential equations [7, 8], fractional order differential equations [9, 10], and phase-field model [11].

Delay integro-differential equations (DIDEs) find extensive applications across various scientific disciplines, such as biosciences, economics, control theory, and material science [12-14]. For these equations, analytical solutions are overly complex, necessitating the use of reliable numerical methods for qualitative analysis. However, efforts to apply DNN to DIDEs and DDEs have been limited. Xing et al in [15] proposed a single-hidden-layer Chebyshev neural network, combined with the extreme learning machine algorithm, to solve delay-integro-differential algebraic equations by dividing the time interval into subintervals, while overlooking the discontinuities due to delay. In the case of integro-differential equations, Lu et al in [2] employs automatic differentiation to compute integer-order derivatives and classical numerical techniques to approximate integral operators. To avoid discretization errors, Yuan et al in [16] replace the integrals in governing equations with auxiliary output variables and employ automatic differentiation of these variables instead of the integral operator.

Multi-task learning (MTL) involves the simultaneous processing of multiple tasks by utilizing a shared learning representation [17]. From an architectural perspective, MTLs are commonly categorized into two types: hard parameter sharing and soft parameter sharing. Hard parameter sharing involves explicitly dividing the parameter set into a shared layer with generic parameters and task-specific layers, which raises the question of which layers should learn the shared information [18]. When different parts of the model share tasks, assuming that sharing is reasonable, those parts of the model become more constrained to good values, which often leads to improved generalization. In contrast to traditional MTL, DNN method for solving PDEs involves explicit equations that describe the relationships between neural network outputs [19, 20]. On the other hand, in traditional MTL, the relationships between tasks are usually unknown and obtained after the training process [21]. Therefore, it is reasonable to utilize MTL techniques when using deep learning methods to solve PDEs and other equations.

In this paper, we propose a DNN method combining MTL and sequential training scheme to solve the forward and inverse problems of DIDEs. We define the auxiliary output variables to represent the integrals in the governing equation, and use the automatic

differentiation of the auxiliary output instead of the integral operator, then divide the time interval based on the delay term, converting the equation into multiple tasks. Subsequently, DNN combined with MTL can be used to incorporate breaking point properties into the loss function, but traditional DNN methods lack these properties. Moreover, to tackle the complexity and optimization issues of the network loss function, we adopt a sequential training scheme. This strategy enables the network to produce the reference solution for the delay term in subsequent tasks, thereby effectively reducing the difficulty of training. To test the effectiveness of different hard parameter sharing structures, we compared the following three parameter sharing structures in numerical experiments: no sharing, full sharing, and partial sharing. The results showed that the partial sharing structure had higher accuracy to some extent than other structures. In addition, we will compare this method with traditional DNN method to demonstrate its superiority. Finally, by training the unknown parameters as hyper-parameters together with the parameters of DNN and adding new residuals of the measurement data to the loss function, we made slight modifications to this method to solve the inverse problem of DIDEs. The numerical simulation results show that even with noisy data, our method can accurately discover unknown parameters in DIDEs.

The remainder of this paper is organized as follows. Section 2 describes the general form of the forward and inverse problems for DIDEs, and the definition of breaking points. In Section 3, we briefly introduce the A-PINN method for DIDEs, subsequently propose the DNNs combining MTL and sequential training scheme for solving DIDEs. In Section 4, we demonstrate the effectiveness of our method for solving the forward and inverse problems of various DIDEs through several examples. Some conclusions and discussions are given in Section 5.

## 2 Delay Integro-Differential Equations

The general form of the initial-value problem for DIDEs can be expressed as

$$\begin{cases} u^{(n)}(t) = f(t, u(t), u(t - \tau)) + \lambda \int_{g(t, \tau)}^{h(t, \tau)} K(t, s)u(s)ds, & t \in [0, T], \\ u(t) = \phi(t), & t \in [-\tau, 0], \end{cases} \quad (2.1)$$

where  $u^{(n)}(t)$  is the  $n$ -order ordinary derivative,  $\tau > 0$  is the time delay,  $f(t, u(t), u(t - \tau))$  is a linear or nonlinear function about  $t, u(t)$  and  $u(t - \tau)$ , besides,  $g(t, \tau)$  and  $h(t, \tau)$  are the bounds of integration,  $K(t, s)$  is the kernel function and  $\phi(t)$  is a given initial function. We define  $t = 0, \tau, \dots, (M - 1)\tau$ , where  $M = \lceil \frac{T}{\tau} \rceil$ , as the breaking points of the equation (2.1), characterized by solutions with weak singularities.

We will consider both forward and inverse problems of DIDEs. In the forward problem, we approximate the solution of  $u(t)$  for any  $t \in [0, T]$ , given the governing equations and initial conditions. The inverse problem occurs when some parameters in the governing

equation are undetermined, and measurement data from  $u(t)$  are employed to identify these parameters.

### 3 Methodology

In this section, we first introduce the application of the A-PINN method to solve DIDEs. We then divide the original equations based on delay and solve them using DNNs with various structures. Finally, a comprehensive explanation of the sequential training scheme is offered.

#### 3.1 A-PINN for Solving DIDEs

We first consider the following DIDE:

$$\begin{cases} u'(t) = f(t, u(t), u(t - \tau)) + \lambda \int_{t-\tau}^t K(s)u(s)ds, & t \in [0, T], \\ u(t) = \phi(t), & t \in [-\tau, 0]. \end{cases} \quad (3.1)$$

Let  $\hat{u}(\boldsymbol{\theta}; t)$  be a fully connected neural network consisting of one input layer,  $L - 1$  hidden layers and one output layer with two outputs. To overcome the limitation of integral discretization, we initially re-express equation (3.1) as follows:

$$\begin{cases} u'(t) = f(t, u(t), u(t - \tau)) + \lambda v(t), & t \in [0, T], \\ v(t) = \int_{t-\tau}^t K(s)u(s)ds, & t \in [0, T], \\ u(t) = \phi(t), & t \in [-\tau, 0], \\ v(0) = \int_{-\tau}^0 K(s)\phi(s)ds. \end{cases} \quad (3.2)$$

Subsequently, we use  $\hat{u}(\boldsymbol{\theta}; t)$  as a surrogate for the solution  $u$  of the equation (3.1) by ensuring that  $\hat{u}$  satisfies the governing equation with the initial function. In addition, we use  $\hat{v}(\boldsymbol{\theta}; t)$  as the auxiliary output of the network to substitute  $v$  in equation (3.1), where  $\boldsymbol{\theta}$  is the set of weights and biases of the network.

The input of the neural network consists of the coordinates of two types of training points:  $\mathcal{N}_i = \{t_1^i, t_2^i, \dots, t_{|\mathcal{N}_i|}^i\}$  and  $\mathcal{N}_f = \{t_1^f, t_2^f, \dots, t_{|\mathcal{N}_f|}^f\}$ . Here,  $t_n^i \in [-\tau, 0]$  ( $n = 1, 2, \dots, |\mathcal{N}_i|$ ) are the initial points and  $t_n^f \in (0, T]$  ( $n = 1, 2, \dots, |\mathcal{N}_f|$ ) are the collocation points for the governing equation.

To ensure the approximate solution conforms to the initial function and governing equation, we substitute  $\hat{u}(\boldsymbol{\theta}; t)$  and  $\hat{v}(\boldsymbol{\theta}; t)$  into equation (3.2). The computed values are then integrated into a term within the loss function  $\mathcal{L}$ , which is defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_i(\boldsymbol{\theta}; \mathcal{N}_i) + \mathcal{L}_f(\boldsymbol{\theta}; \mathcal{N}_f) + \mathcal{L}_a(\boldsymbol{\theta}; \mathcal{N}_f). \quad (3.3)$$

In the expression above,

$$\mathcal{L}_i(\boldsymbol{\theta}; \mathcal{N}_i) = \frac{1}{|\mathcal{N}_i|} \sum_{t \in \mathcal{N}_i} |\hat{u}(\boldsymbol{\theta}; t) - \phi(t)|^2$$

denotes the loss term for the initial time interval  $[-\tau, 0]$ ,

$$\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{N}_f) = \frac{1}{|\mathcal{N}_f|} \sum_{t \in \mathcal{N}_f} \left| \frac{\partial \hat{u}(\boldsymbol{\theta}; t)}{\partial t} - f(t, \hat{u}(\boldsymbol{\theta}; t), \hat{u}(\boldsymbol{\theta}; t - \tau)) - \lambda v(\boldsymbol{\theta}, t) \right|^2$$

is the loss term of the governing equation calculated in  $(0, T]$ , and

$$\mathcal{L}_a(\boldsymbol{\theta}; \mathcal{N}_f) = \frac{1}{|\mathcal{N}_f|} \sum_{t \in \mathcal{N}_f} \left| \frac{\partial \hat{v}(\boldsymbol{\theta}; t)}{\partial t} - K(t)\hat{u}(\boldsymbol{\theta}; t) + K(t - \tau)\hat{u}(\boldsymbol{\theta}; t - \tau) \right|^2$$

denotes the loss term for the auxiliary output  $v(t)$ , with the relation of  $dv(t)/dt = K(t)u(t) - K(t - \tau)u(t - \tau)$ .

Finally, the neural network is optimized to identify the optimal  $\boldsymbol{\theta}^*$  by minimizing the loss function  $\mathcal{L}(\boldsymbol{\theta})$ . Due to the highly nonlinear and nonconvex characteristics of the loss function, gradient-based optimization algorithms like Adam, gradient descent, and L-BFGS are typically utilized to optimize the loss function.

The derivative of the network output with respect to the input  $t$  can be computed using the automatic differentiation (AD) technique [22]. This technique is also applicable for the convenient computation of all required gradients with respect to the network parameters  $\boldsymbol{\theta}$ . Unlike classical numerical differential methods such as finite difference, AD can produce exact partial derivatives without truncation error [2]. Furthermore, AD can be easily implemented on widely-used deep learning platforms such as TensorFlow and PyTorch, so we use PyTorch and utilize the L-BFGS optimizer in this paper. Figure 1 shows the detailed schematic for solving DIDEs using PINN.

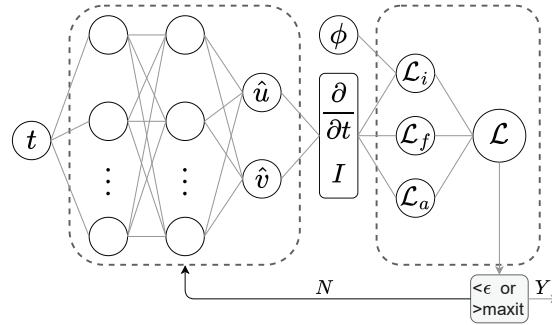


Figure 1 Schematic structure of A-PINN for solving DIDEs.

To discover the unknown parameters in the equations using measurement data, such as the parameter  $\lambda$ , a fine-tuning of the A-PINN framework is sufficient. This involves incorporating  $\lambda$  into the neural network, which is then trained with  $\boldsymbol{\theta}$ . In this scenario, the loss function related to the measurement data is included in the overall loss function

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}, \lambda_{inv}) = & \mathcal{L}_i(\boldsymbol{\theta}, \lambda_{inv}; \mathcal{N}_i) + \mathcal{L}_f(\boldsymbol{\theta}, \lambda_{inv}; \mathcal{N}_f) \\ & + \mathcal{L}_a(\boldsymbol{\theta}, \lambda_{inv}; \mathcal{N}_f) + \mathcal{L}_{inv}(\boldsymbol{\theta}, \lambda_{inv}; \mathcal{N}_{inv}), \end{aligned} \quad (3.4)$$

where

$$\mathcal{L}_{inv}(\boldsymbol{\theta}, \lambda_{inv}; \mathcal{N}_{inv}) = \frac{1}{|\mathcal{N}_{inv}|} \sum_{t \in \mathcal{N}_{inv}} |\hat{u}(\boldsymbol{\theta}, \lambda_{inv}; t) - u^{inv}(t)|^2.$$

Here,  $\mathcal{N}_{inv}$  represents the set of measurement points within the time interval,  $u^{inv}(t)$  denotes the measurement data, and  $\lambda_{inv}$  represents the unknown parameter in the DIDE. Noting that the first three terms in (3.4) correspond to those in (3.3), and the last one includes the trainable parameter.

### 3.2 DNNs with Different Parameter Sharing Structures

In this section, we utilize DNNs with diverse sharing structures to solve the DIDEs. Similarly to A-PINN, we first transform equation (3.1) into the form (3.2) and subsequently by dividing the time interval  $[0, T]$  into subintervals  $T_1 = [0, \tau], T_2 = [\tau, 2\tau], \dots$  and  $T_M = [(M-1)\tau, T]$ , where  $M = \lceil \frac{T}{\tau} \rceil$ , the original equation can be expressed as

$$\begin{cases} u'_1(t) = f(t, u_1(t), u_1(t-\tau)) + \lambda v_1(t), & t \in [0, T], \\ v_1(t) = \int_{t-\tau}^t K(s)u_1(s)ds, & t \in [0, T], \\ u_1(t) = \phi(t), & t \in [-\tau, 0], \\ v_1(0) = \int_{-\tau}^0 K(s)\phi(s)ds \end{cases} \quad (3.5)$$

for the subinterval  $T_1$ , and

$$\begin{cases} u'_m(t) = f(t, u_m(t), u_m(t-\tau)) + \lambda v_m(t), & t \in [0, T], \\ v_m(t) = \int_{t-\tau}^t K(s)u_m(s)ds, & t \in [0, T], \\ u_m(t) = u_{m-1}(t), & t \in [-\tau, 0], \\ v_m((m-1)\tau) = v_{m-1}((m-1)\tau) \end{cases} \quad (3.6)$$

for the subintervals  $T_m$  ( $m = 2, \dots, M$ ).

Regarding parameter sharing in networks, we consider three representative sharing structures: no sharing, partial sharing, and full sharing. Then we construct a neural network with  $2M$  outputs and present five network structures. Figure 2 illustrates schematic diagrams of the various sharing structures for  $M = 3$ . The first three structures are partial sharing structures, each comprising generic parameters and task-specific parameters. Here,  $\hat{u}_m(\boldsymbol{\theta}; t^m)$  and  $\hat{v}_m(\boldsymbol{\theta}; t^m)$  ( $t^m \in T_m$ ) denote outputs used to approximate the solution  $u$  and the integral  $v$  within the subinterval  $T_m$ , respectively. The accuracy of the various network structures will be compared in each numerical experiment in Section 4, and the specific architectures of these structures will also be presented in this section.

The loss function over all subintervals is defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{m=1}^M \left[ \mathcal{L}_d^m(\boldsymbol{\theta}; \mathcal{N}_d^m) + \mathcal{L}_f^m(\boldsymbol{\theta}; \mathcal{N}_f^m) + \mathcal{L}_{da}^m(\boldsymbol{\theta}; \mathcal{N}_d^m) + \mathcal{L}_a^m(\boldsymbol{\theta}; \mathcal{N}_f^m) \right]$$

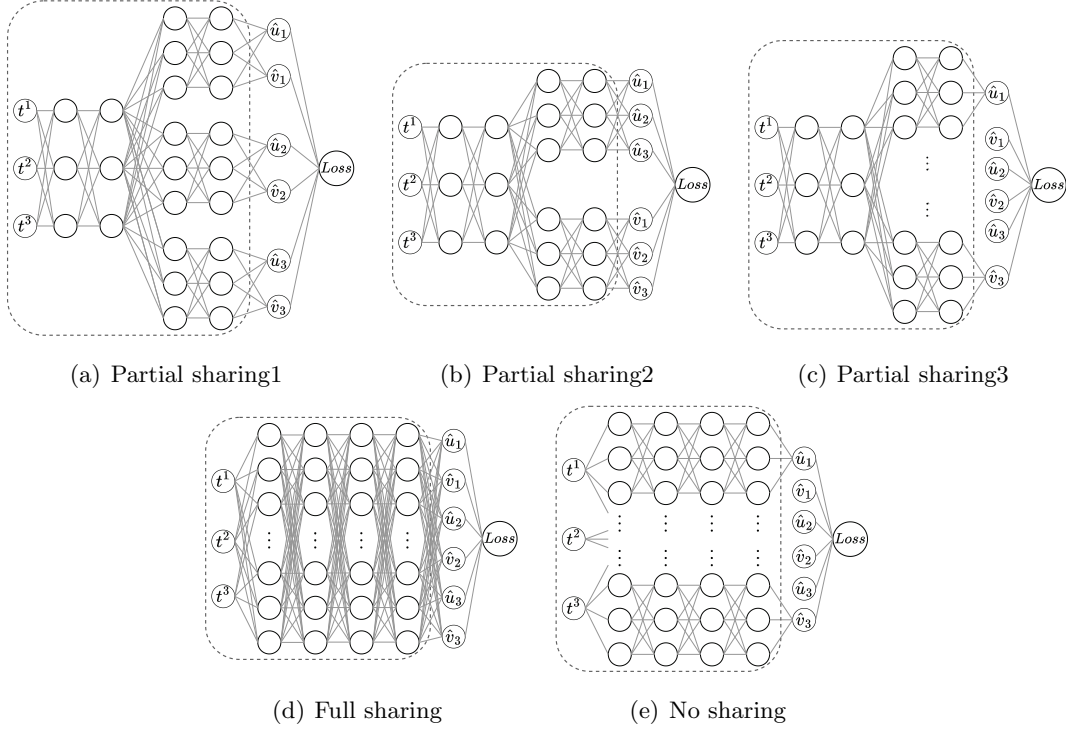


Figure 2 Schematic diagrams of networks with different sharing structures.

where

$$\mathcal{L}_d^m(\boldsymbol{\theta}; \mathcal{N}_d^m) = \sum_{i=1}^m \left[ \left. \frac{\partial^{(i-1)} \hat{u}_m(\boldsymbol{\theta}; t)}{\partial t^{(i-1)}} \right|_{t=k-1} - \left. \frac{\partial^{(i-1)} \hat{u}_{m-1}(\boldsymbol{\theta}; t)}{\partial t^{(i-1)}} \right|_{t=k-1} \right]^2 \quad (3.7)$$

is the loss term of  $u(t)$  at the  $m$ -th breaking point,

$$\mathcal{L}_f^m(\boldsymbol{\theta}; \mathcal{N}_f^m) = \frac{1}{|\mathcal{N}_f^m|} \sum_{t \in \mathcal{N}_f^m} \left| \frac{\partial \hat{u}_m(\boldsymbol{\theta}; t)}{\partial t} - f(t, \hat{u}_m(\boldsymbol{\theta}; t), \hat{u}_{m-1}(\boldsymbol{\theta}; t - \tau)) - \lambda v_m(\boldsymbol{\theta}, t) \right|^2$$

is the loss term of the governing equation on the  $m$ -th subinterval,

$$\mathcal{L}_{da}^m(\boldsymbol{\theta}; \mathcal{N}_d^m) = |\hat{v}_m(\boldsymbol{\theta}; (m-1)\tau) - \hat{v}_{m-1}(\boldsymbol{\theta}; (m-1)\tau)|^2$$

denotes the loss term of the auxiliary output  $v(t)$  at the  $m$ -th breaking point and

$$\mathcal{L}_a^m(\boldsymbol{\theta}; \mathcal{N}_f^m) = \frac{1}{|\mathcal{N}_f^m|} \sum_{t \in \mathcal{N}_f^m} \left| \frac{\partial \hat{v}_m(\boldsymbol{\theta}; t)}{\partial t} - K(t) \hat{u}_m(\boldsymbol{\theta}; t) + K(t - \tau) \hat{u}_m(\boldsymbol{\theta}; t - \tau) \right|^2$$

is the loss term of the auxiliary output  $v(t)$  on the  $m$ -th subinterval. Here  $\mathcal{N}_f^m$  denotes the set of collocation points within the  $m$ -th subinterval, and  $\mathcal{N}_d^m$  represents the set of the  $m$ -th breaking point. Specifically, we have  $\hat{u}_0(\boldsymbol{\theta}; t) = \phi(t)$ ,  $\hat{v}_0(\boldsymbol{\theta}; 0) = \int_{-\tau}^0 K(s) \phi(s) ds$ . It is

important to note that  $(i - 1)$  in (3.7) represents the subscript  $(i - 1)$ -th order derivative of  $\hat{u}_m(\boldsymbol{\theta}; t)$  with respect to  $t$ .

The inclusion of these loss function terms and the integration of derivative information is motivated by the regularity of the exact solution at the breaking points. In particular, the DIDE (3.1) generally demonstrates  $C^{m-1}$  regularity at the breaking point  $t = (m - 1)\tau$  [23, 24]. The specific form of the loss function  $\mathcal{L}_d^m$  varies among different equations and is constructed by the properties at the breaking points, which may not always involve derivatives.

After training, the final solution is obtained as the concatenated set of  $M$  outputs from the neural network  $\hat{u}(\boldsymbol{\theta}; t)$ :

$$u_{pred} = \bigcup_{m=1}^M \hat{u}_m(\boldsymbol{\theta}; t^m), \quad t^m \in T_m. \quad (3.8)$$

### 3.3 Sequential Training Scheme

Overly complex loss functions can increase the difficulty of network optimization, such as the additional tasks and outputs introduced by MTL. Moreover, the training process is significantly complicated by the influence of the previous state  $\hat{u}_{m-1}(\boldsymbol{\theta}; t^m - \tau)$  on the solution  $\hat{u}_m(\boldsymbol{\theta}; t^m)$  of the DIDE. To address the aforementioned difficulties, we employ a sequential training scheme (STS) in which the approximation of the previous task serves as a reference solution for the delay term in the subsequent task.

In STS, the outputs  $\hat{u}_m(\hat{\boldsymbol{\theta}}; t^m)$  and  $\hat{v}_m(\hat{\boldsymbol{\theta}}; t^m)$  on the subinterval  $T_m$  are considered as the  $m$ -th task. Subsequently, the first  $m$  tasks are trained sequentially using the equations (3.5) and (3.6) to obtain a reference solution for the delay term in the next task. In the following, we begin with computation of the reference solution on the interval  $T_1$ . The corresponding loss function for problem (3.5) can be defined as:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_d^1(\boldsymbol{\theta}; \mathcal{N}_d^1) + \mathcal{L}_f^1(\boldsymbol{\theta}; \mathcal{N}_f^1) + \mathcal{L}_{da}^1(\boldsymbol{\theta}; \mathcal{N}_d^1) + \mathcal{L}_a^1(\boldsymbol{\theta}; \mathcal{N}_f^1)$$

where

$$\begin{aligned} \mathcal{L}_d^1(\boldsymbol{\theta}; \mathcal{N}_d^1) &= |\hat{u}_1(\boldsymbol{\theta}; 0) - \phi(0)|^2, \\ \mathcal{L}_{da}^1(\boldsymbol{\theta}; \mathcal{N}_d^1) &= \left| \hat{v}_1(\boldsymbol{\theta}; 0) - \int_{-\tau}^0 K(s)\phi(s)ds \right|^2 \end{aligned}$$

are the loss terms of  $u(t)$  and auxiliary output  $v(t)$  at the first breaking point  $t = 0$ , and

$$\begin{aligned} \mathcal{L}_f^1(\boldsymbol{\theta}; \mathcal{N}_f^1) &= \frac{1}{|\mathcal{N}_f^1|} \sum_{t \in \mathcal{N}_f^1} \left| \frac{\partial \hat{u}_1(\boldsymbol{\theta}; t)}{\partial t} - f(t, \hat{u}_1(\boldsymbol{\theta}; t), \phi(t - \tau)) - \lambda v_1(\boldsymbol{\theta}, t) \right|^2, \\ \mathcal{L}_a^1(\boldsymbol{\theta}; \mathcal{N}_f^1) &= \frac{1}{|\mathcal{N}_f^1|} \sum_{t \in \mathcal{N}_f^1} \left| \frac{\partial \hat{v}_1(\boldsymbol{\theta}; t)}{\partial t} - K(t)\hat{u}_1(\boldsymbol{\theta}; t) + K(t - \tau)\phi(t - \tau) \right|^2 \end{aligned}$$

are the loss terms of the governing equation and the auxiliary output  $v(t)$  on the first subinterval  $T_1$ . Here  $\mathcal{N}_f^1$  denotes the set of collocation points on the first subinterval, and  $\mathcal{N}_d^1 = \{0\}$  represents the set of the first breaking point. After a specified training step or when the loss function falls below a predetermined threshold, the outputs  $\hat{u}_1(\boldsymbol{\theta}; t^1)$  and  $\hat{v}_1(\boldsymbol{\theta}; t^1)$  are saved as the reference solution  $\hat{u}_1^*(t^1)$  and  $\hat{v}_1^*(t^1)$ . These reference solutions will be utilized as initial functions for the subsequent task training.

Next, assuming that the reference solution  $\hat{u}_1^*(t^1), \hat{u}_2^*(t^2), \dots, \hat{u}_{m-1}^*(t^{m-1})$  and  $\hat{v}_1^*(t^1), \hat{v}_2^*(t^2), \dots, \hat{v}_{m-1}^*(t^{m-1})$  on the subinterval  $T_1, T_2, \dots, T_{m-1}$  have been obtained, the loss function for the first  $m$  subintervals is defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{k=1}^m \left[ \mathcal{L}_{d1}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) + \mathcal{L}_f^k(\boldsymbol{\theta}; \mathcal{N}_f^k) + \mathcal{L}_{da}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) + \mathcal{L}_a^k(\boldsymbol{\theta}; \mathcal{N}_f^k) \right] + \sum_{k=2}^m \mathcal{L}_{d2}^k(\boldsymbol{\theta}; \mathcal{N}_d^k),$$

where

$$\begin{aligned} \mathcal{L}_{d1}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) &= |\hat{u}_k(\boldsymbol{\theta}; (k-1)\tau) - \hat{u}_{k-1}^*((k-1)\tau)|^2, \\ \mathcal{L}_{d2}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) &= \left[ \frac{\partial^{(i-1)} \hat{u}_k(\boldsymbol{\theta}; t)}{\partial t^{(i-1)}} \Big|_{t=k-1} - \frac{\partial^{(i-1)} \hat{u}_{k-1}(\boldsymbol{\theta}; t)}{\partial t^{(i-1)}} \Big|_{t=k-1} \right]^2, \\ \mathcal{L}_{da}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) &= |\hat{v}_k(\boldsymbol{\theta}; (k-1)\tau) - \hat{v}_{k-1}^*((k-1)\tau)|^2 \end{aligned}$$

are the loss terms of  $u(t)$  and  $v(t)$  at the  $k$ -th breaking point, and

$$\begin{aligned} \mathcal{L}_f^k(\boldsymbol{\theta}; \mathcal{N}_f^k) &= \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \frac{\partial \hat{u}_k(\boldsymbol{\theta}; t)}{\partial t} - f(t, \hat{u}_k(\boldsymbol{\theta}; t), \hat{u}_{k-1}^*(t-\tau)) - \lambda v_k(\boldsymbol{\theta}, t) \right|^2, \\ \mathcal{L}_a^k(\boldsymbol{\theta}; \mathcal{N}_f^k) &= \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \frac{\partial \hat{v}_k(\boldsymbol{\theta}; t)}{\partial t} - K(t) \hat{u}_k(\boldsymbol{\theta}; t) + K(t-\tau) \hat{u}_{k-1}^*(t-\tau) \right|^2 \end{aligned}$$

are the loss terms of the governing equation and the auxiliary output on the  $k$ -th subinterval. Here  $\mathcal{N}_f^k$  represents the set of training points on the  $k$ -th subinterval, while  $\mathcal{N}_d^k$  denotes the set of the  $k$ -th breaking point. Specifically,  $\hat{u}_0^*(t) = \phi(t)$ ,  $\hat{v}_0^*(0) = \int_{-\tau}^0 K(s) \phi(s) ds$ .

Upon completion of training for the first  $m$  tasks,  $\hat{u}_k(\boldsymbol{\theta}; t^k), \hat{v}_k(\boldsymbol{\theta}; t^k), k = 1, 2, \dots, m$  are saved as reference solutions  $\hat{u}_k^*(t^k), \hat{v}_k^*(t^k)$ . Subsequently, the ultimate solution is obtained using (3.8) after sequential training.

## 4 Numerical Experiments

This section utilizes DNNs with various parameter sharing structures in conjunction with STS to solve different DIDEs, showcasing the effectiveness of the method. Next, we provide the training conditions for the numerical examples. All examples are trained on a GeForce RTX 4090 GPU using the *tanh* activation function and the L-BFGS optimization algorithm. The fixed learning rate employed is 0.01. Latin hypercube sampling is employed to select the training points [25]. All algorithms are coded in Python with PyTorch.

In each numerical example of the forward problem, we initially compare the effectiveness of the five network structures discussed in Section 3.2, followed by a comparison with A-PINN using the most effective network that incorporates STS. In the following,  $1 - N_g - N_g / - N_t - N_t - 2(*3)$  denotes a DNN with three tasks, the generic parameters consist of two layers, each containing  $N_g$  neurons, while the task-specific parameters consist of three parts, each with two layers of  $N_t$  neurons and two outputs. To assess the accuracy of the solutions, the relative error (RE) is calculated using the formula  $\|u_{exact} - u_{pred}\|_2 / \|u_{exact}\|_2$ . Here,  $\|\cdot\|_2$  represents the  $L_2$  norm,  $u_{exact}$  and  $u_{pred}$  correspond to the exact solution and the predicted solution obtained from the neural network, respectively.

#### 4.1 The Forward Problem of DIDE

In this subsection we solve the following simple DIDE

$$\begin{cases} u'(t) = \lambda u(t-1) + \mu \int_{t-1}^t u(s) ds, & t \in [0, 3], \\ u(t) = 1, & t \in [-1, 0]. \end{cases} \quad (4.1)$$

Firstly, we induce an auxiliary output  $v(t)$  to represent the integral, and rewrite equation (4.1) as

$$\begin{cases} u'(t) = \lambda u(t-1) + \mu v(t), & t \in [0, 3], \\ v(t) = \int_{t-1}^t u(s) ds, & t \in [0, 3], \\ u(t) = 1, & t \in [-1, 0], \\ v(t) = 1, & t \in [-1, 0]. \end{cases}$$

Subsequently, by dividing the time interval  $[0, 3]$  into three subintervals based on  $\tau = 1$ , and assuming that we have obtained the reference solutions for the first two subintervals, we define the corresponding loss function over all three subintervals as

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{k=1}^3 \left[ \mathcal{L}_{d1}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) + \mathcal{L}_f^k(\boldsymbol{\theta}; \mathcal{N}_f^k) + \mathcal{L}_{da}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) + \mathcal{L}_a^k(\boldsymbol{\theta}; \mathcal{N}_f^k) \right] + \sum_{k=2}^3 \mathcal{L}_{d2}^k(\boldsymbol{\theta}; \mathcal{N}_d^k),$$

where  $\mathcal{L}_{d1}$ ,  $\mathcal{L}_{d2}$  and  $\mathcal{L}_{da}$  are derived from subsection 3.3 at the special case of  $\tau = 1$ . Since the DIDE (4.1) exhibits  $C^{k-1}$  regularity at the breaking point  $t = (k-1)\tau = k-1$  (see [25]).

$$\begin{aligned} \mathcal{L}_f^k(\boldsymbol{\theta}; \mathcal{N}_f^k) &= \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \frac{\partial \hat{u}_k(\boldsymbol{\theta}; t)}{\partial t} - \lambda \hat{u}_{k-1}^*(t-1) - \mu v_k(\boldsymbol{\theta}, t) \right|^2, \\ \mathcal{L}_a^k(\boldsymbol{\theta}; \mathcal{N}_f^k) &= \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \frac{\partial \hat{v}_k(\boldsymbol{\theta}; t)}{\partial t} - \hat{u}_k(\boldsymbol{\theta}; t) + \hat{u}_{k-1}^*(t-1) \right|^2 \end{aligned}$$

are the loss terms of the governing equation and the auxiliary output on the  $k$ -th subinterval. Specifically,  $\hat{u}_0^*(t) = 1$ ,  $\hat{v}_0^*(0) = 1$ .

Let  $\lambda = 1, \mu = 1$ , we can obtain the exact solution of equation (4.1) as

$$u(t) = \begin{cases} e^t - e^{-t} + 1, & t \in [0, 1), \\ -te^{-t+1} + \frac{1}{2}e^{t-1} + \frac{1}{2}e^{-t+1} + e^t - e^{-t} + 1, & t \in [1, 2), \\ \frac{-2t^2+6t-5}{4}e^{-t+2} + \frac{-2t+1}{2}e^{-t+1} + e^t - e^{-t} + \frac{e^{t-2}}{4} + \frac{e^{t-1}}{2} + 1, & t \in [2, 3]. \end{cases}$$

The exact solutions for other parameters of the equation can also be obtained.

Table 1 displays the configurations for five testing parameter sharing structures, all having a nearly identical total number of network parameters, where L represents the number of layers and N represents the number of neurons per layer. After solving equation (4.1) with  $\lambda = 1, \mu = 1$  using the five tests and STS, Table 2 offers a quantitative comparison of the results, with bolded loss values and RE highlighting the optimal results. The networks with full sharing and no sharing structure performed worse than the referential test, but the no sharing structure resulted in the shortest training time.

Table 1 Five testing structures used in subsections 4.1 and 4.4.

Test name	Structure	Layers	Layers	Number of Parameters
		shared	Independent	
Partial sharing1	1-30-30/-30-30-2(*3)	2L*30N	2L*30N	6756
Partial sharing2	1-30-30/-39-39-3(*2)	2L*30N	2L*39N	6768
Partial sharing3	1-30-30/-19-19-1(*6)	2L*30N	2L*19N	6924
Full sharing	1-46-46-46-46-6	4L*46N	0L	6860
No sharing	1-23-23-23-23-1(*6)	0L	4L*23N	7044

Next, we utilize the structure with the smallest relative error: partial sharing2 with STS, to solve equation (4.1) with various Parameters. Subsequently, it will be compared with A-PINN and partial sharing2 without STS.

For A-PINN, the numbers of collocation points on  $[-1, 0]$  and  $[0, 3]$  are  $|\mathcal{N}_i| = 50$  and  $|\mathcal{N}_f| = 153$ , respectively, and the network architecture is  $1 - 47 - 47 - 47 - 47 - 2$  with a total of 6958 parameters. For partial sharing2 with or without STS, we set the numbers of collocation points  $|\mathcal{N}_f^m| = 50, m = 1, 2, 3$  for each subinterval and the breaking points are  $\mathcal{N}_d^1 = \{0\}, \mathcal{N}_d^2 = \{1\}, \mathcal{N}_d^3 = \{2\}$ . Each scheme undergoes 500 iterations, with STS involved in the second and third tasks at the 100th and 250th iterations, respectively.

Figure 3 compares the loss functions during the training of the three schemes used to solve (4.1). The loss function of partial sharing2 with STS is minimal, indicating a less arduous optimization challenge for equation (4.1) with varying Parameters, and it sharply

Table 2 Comparison of testing results among the five structures in Table 1 for solving DIDEs.

Term	Test name	Iteration-300	Iteration-400	Iteration-500	Time
Loss	Partial sharing1	1.84e-3	<b>5.21e-5</b>	<b>4.49e-5</b>	8.74s
	Partial sharing2	5.35e-4	1.03e-4	7.07e-5	9.72s
	Partial sharing3	<b>2.57e-4</b>	5.91e-5	5.89e-5	9.79s
	Full sharing	5.91e-1	3.59e-4	2.03e-4	8.84s
	No sharing	3.82e-4	7.52e-5	6.76e-5	<b>8.68s</b>
RE	Partial sharing1	1.75e-4	2.59e-5	2.50e-5	
	Partial sharing2	<b>4.59e-5</b>	2.15e-5	<b>1.70e-5</b>	
	Partial sharing3	1.06e-4	<b>2.11e-5</b>	1.98e-5	
	Full sharing	4.13e-2	1.01e-4	7.06e-5	
	No sharing	8.33e-4	5.24e-5	5.13e-5	

increases with the addition of each subsequent training task. Notably, partial sharing2 without STS fails to converge for Parameters  $\lambda = 5$  and  $\mu = -3$ , and thus is not depicted in the figures or table. Figure 4 compares the prediction results of various methods against the exact solution at different equation parameters, along with the absolute values of point-wise errors. Partial sharing2, with or without STS, yields good results in cases of convergence. However, the error without STS gradually increases over time, while the addition of STS has to some extent reduced this problem. Table 3 shows the relative errors and training times for the three schemes. Due to A-PINN not taking into account the information at the delay points, the relative errors of partial sharing2 with STS are 2-3 orders of magnitude higher than those of A-PINN. It can be found that STS effectively reduces training difficulty, decreases training time, and enhances approximation accuracy across various parameters. Additionally, for equations with different weak singularities at the breaking points, partial sharing2 exhibits high convergence difficulty in certain cases and may even fail to converge, highlighting the necessity of STS.

## 4.2 The Forward Problem of NDIDE

In this subsection, we examine the following neutral delay integro-differential equation (NDIDE) with a discontinuous initial function:

$$\begin{cases} \frac{d}{dt}[u(t) - u(t-1)] = \lambda u(t-1) + \mu \int_{t-1}^t u(s) ds & t \in [0, 2], \\ u(t) = \phi(t) & t \in [-1, 0], \end{cases} \quad (4.2)$$

where  $\phi(t) = 1$  for  $t < 0$  and  $\phi(0) = 2$ .

In NDIDEs, the evolution of phenomena is influenced by both the delay term and its

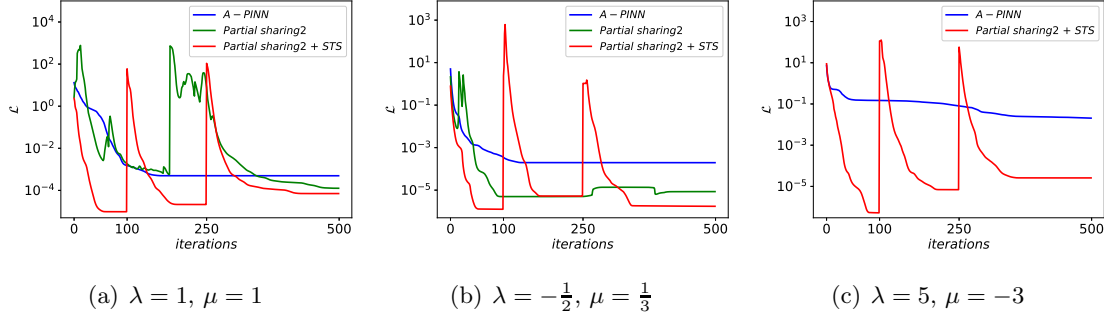


Figure 3 Loss history of diverse schemes for DIDEs with different parameters.

Table 3 Comparison of diverse schemes for solving DIDEs with different parameters.

Parameters	NN structure	STS	RE	Loss	Time
$\lambda = 1, \mu = 1$	A-PINN		1.53e-3	5.42e-4	12.1s
	Partial sharing2		8.44e-5	1.26e-4	15.6s
	Partial sharing2	✓	1.70e-5	7.07e-5	9.72s
$\lambda = -\frac{1}{2}, \mu = \frac{1}{3}$	A-PINN		1.47e-2	1.96e-5	9.41s
	Partial sharing2		2.08e-4	7.59e-6	7.68s
	Partial sharing2	✓	9.84e-5	1.73e-6	5.56s
$\lambda = 5, \mu = -3$	A-PINN		2.11e-2	2.05e-3	27.2s
	Partial sharing2		–	–	–
	Partial sharing2	✓	7.38e-5	1.73e-6	8.18s

derivative. NDIDEs produce discontinuities determined by the delay function when the initial function and the exact solution are not seamlessly connected. These discontinuities disrupt the overall smoothness of the solution, making it more challenging to achieve precise numerical results.

Similar to the approach used for solving DIDEs, we first rewrite equation (4.2) as the following delay integro-differential algebraic equation:

$$\left\{ \begin{array}{l} \frac{d}{dt}[w(t)] = \lambda u(t-1) + \mu v(t), \quad t \in [0, 2], \\ v(t) = \int_{t-1}^t u(s) ds, \quad t \in [0, 2], \\ w(t) = u(t) - u(t-1), \quad t \in [0, 2], \\ u(t) = \phi(t), \quad t \in [-1, 0], \\ v(t) = 1, \quad t \in [-1, 0], \\ w(0) = 1. \end{array} \right.$$

Next, we divide the time interval  $[0, 2]$  into two subintervals  $T_1 = [0, 1], T_2 = [1, 2]$ , and then assuming that the reference solutions  $\hat{u}_1^*(t^1)$ ,  $\hat{v}_1^*(t^1)$ , and  $\hat{w}_1^*(t^1)$  on the first subinterval

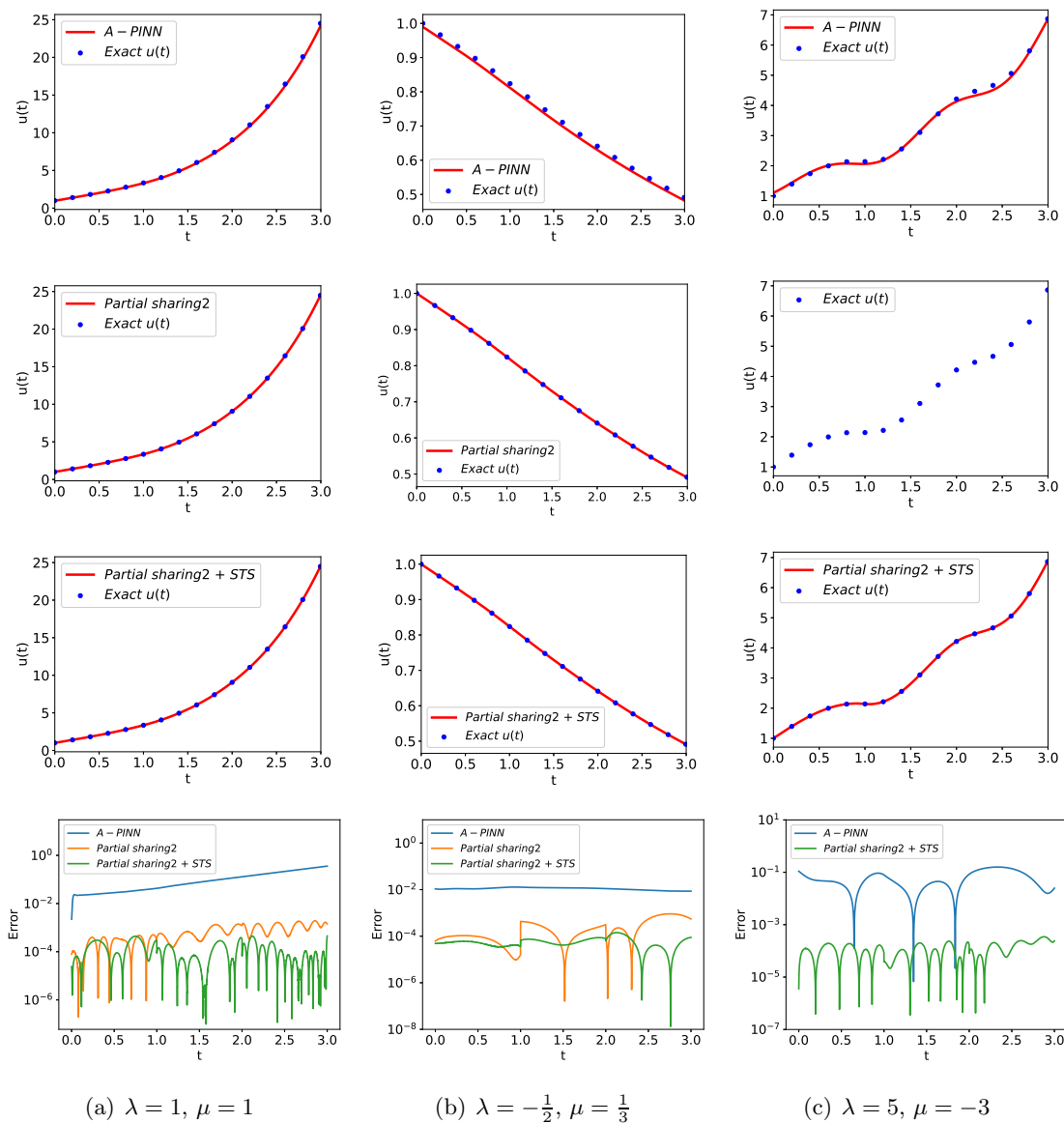


Figure 4 The exact solution, predicted solutions and the absolute value of point-wise errors of diverse schemes for DIDEs with different parameters.

have been obtained, the loss function is defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{k=1}^2 \left[ \mathcal{L}_f^k(\boldsymbol{\theta}; \mathcal{N}_f^k) + \mathcal{L}_{da1}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) + \mathcal{L}_{a1}^k(\boldsymbol{\theta}; \mathcal{N}_f^k) + \mathcal{L}_{da2}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) + \mathcal{L}_{a2}^k(\boldsymbol{\theta}; \mathcal{N}_f^k) \right],$$

where

$$\mathcal{L}_f^k(\boldsymbol{\theta}; \mathcal{N}_f^k) = \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \frac{\partial \hat{w}_k(\boldsymbol{\theta}; t)}{\partial t} - \lambda \hat{u}_{k-1}^*(t-1) - \mu v_k(\boldsymbol{\theta}, t) \right|^2$$

is the loss term of the governing equation on the  $k$ -th subinterval,

$$\mathcal{L}_{da1}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) = \left| \hat{v}_k(\boldsymbol{\theta}; k-1) - \hat{v}_{k-1}^*(k-1) \right|^2,$$

$$\mathcal{L}_{da2}^k(\boldsymbol{\theta}; \mathcal{N}_d^k) = \left| \hat{w}_k(\boldsymbol{\theta}; k-1) - \hat{w}_{k-1}^*(k-1) \right|^2$$

denote the loss terms of the auxiliary outputs  $v(t)$  and  $w(t)$  at the  $k$ -th breaking point, according to the property of the exact solution, that is,  $v(t)$  is continuous on the time interval.

$$\mathcal{L}_{a1}^k(\boldsymbol{\theta}; \mathcal{N}_f^k) = \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \frac{\partial \hat{v}_k(\boldsymbol{\theta}; t)}{\partial t} - \hat{u}_k(\boldsymbol{\theta}; t) + \hat{u}_{k-1}^*(t-1) \right|^2,$$

$$\mathcal{L}_{a2}^k(\boldsymbol{\theta}; \mathcal{N}_f^k) = \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \hat{w}_k(\hat{\boldsymbol{\theta}}; t) - \hat{u}_k(\hat{\boldsymbol{\theta}}; t) + \hat{u}_{k-1}^*(t-1) \right|^2$$

are the loss terms of the auxiliary outputs  $v(t)$  and  $w(t)$  on the  $k$ -th subinterval. Specifically,  $\hat{u}_0^*(t) = 1$ ,  $\hat{v}_0^*(0) = 1$  and  $\hat{w}_0^*(0) = 1$ .

Table 4 presents the configurations of the five testing parameter sharing structures employed to solve the NDIDE. Table 5 provides a quantitative comparison of the solutions to the equation (4.2) with  $\lambda = 1$ ,  $\mu = 1$  utilizing these five structures. Similar to the results obtained from solving equation (4.1), the loss functions and relative errors of the partial sharing structures are smaller than those of the no sharing and full sharing structures. Notably, partial sharing2 exhibits the smallest loss function, while partial sharing3 shows the smallest relative error.

Next we use partial sharing3 for comparison with A-PINN. For A-PINN, the numbers of collocation points on  $[-1, 0]$  and  $[0, 2]$  are  $|\mathcal{N}_i| = 50$  and  $|\mathcal{N}_f| = 102$ , respectively, and the network architecture is  $1 - 47 - 47 - 47 - 47 - 3$  with a total of 7006 parameters. For partial sharing3 with or without STS, we set the numbers of collocation points  $|\mathcal{N}_f^m| = 50$ ,  $m = 1, 2$  for each subinterval and the breaking points are  $\mathcal{N}_d^1 = \{0\}$ ,  $\mathcal{N}_d^2 = \{1\}$ . Each of the three schemes undergoes 500 iterations, with STS engaging in the second task at the 200th iteration.

Figure 5 illustrates a comparison of the loss functions during the training of the three schemes used to solve (4.2). It is evident that partial sharing3 is more challenging to train

Table 4 Five testing structures used in subsection 4.2.

Test name	Structure	Layers shared	Layers Independent	Number of Parameters
Partial sharing1	1-30-30/-40-40-3(*2)	2L*30N	2L*40N	6996
Partial sharing2	1-30-30/-31-31-2(*3)	2L*30N	2L*31N	7041
Partial sharing3	1-30-30/-19-19-1(*6)	2L*30N	2L*19N	6924
Full sharing	1-46-46-46-46-6	4L*46N	0L	6860
No sharing	1-23-23-23-23-1(*6)	0L	4L*23N	7044

Table 5 Comparison of testing results among the five structures in Table 4 for solving NDIDE.

Term	Test name	Iteration-300	Iteration-400	Iteration-500	Time
	Partial sharing1	6.02e-5	4.92e-5	4.91e-5	6.65s
	Partial sharing2	1.07e-4	<b>2.27e-5</b>	<b>2.26e-5</b>	7.42s
Loss	Partial sharing3	<b>5.78e-5</b>	3.88e-5	3.84e-5	6.33s
	Full sharing	1.19e-3	4.11e-4	4.08e-4	7.62s
	No sharing	1.31e-4	7.12e-5	7.11e-5	<b>6.17s</b>
	Partial sharing1	<b>3.53e-4</b>	2.80e-4	2.77e-4	
	Partial sharing2	5.31e-4	2.96e-5	2.91e-4	
RE	Partial sharing3	3.97e-4	<b>2.19e-4</b>	<b>2.12e-4</b>	
	Full sharing	1.70e-3	9.44e-4	8.58e-4	
	No sharing	7.39e-4	6.17e-4	5.82e-4	

without STS compared to when STS is included. Furthermore, the loss function of partial sharing3 with STS the smallest among all three sets of parameters. Figure 6 presents a comparison of the prediction results from various methods against the exact solution at different equation parameters, along with the absolute values of point-wise errors. A-PINN exhibits a significant error at the breaking points, which is mitigated by partial sharing3 and further improved with STS. However, when  $\lambda = \frac{9}{2}$ ,  $\mu = -\frac{5}{2}$ , partial sharing3 without STS incurs a greater loss and exhibits a higher error, whereas incorporating STS resolves this issue, indicating that the nature of the equations influences the optimization difficulty.. Table 6 provides the relative errors and training times associated with the three schemes. The relative errors of partial sharing3 with STS are two orders of magnitude higher than those of A-PINN.

### 4.3 The Forward Problem of Nonlinear Partial DIDE

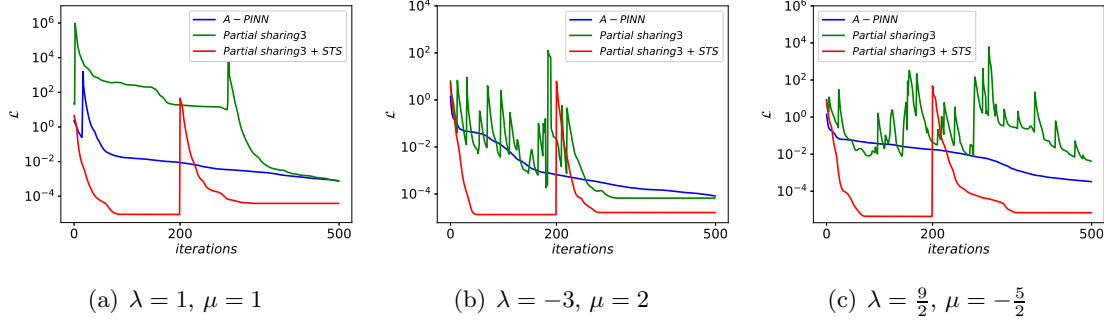


Figure 5 Loss history of diverse schemes for NDIDEs with different parameters.

Table 6 Comparison of diverse schemes for solving NDIDEs with different parameters.

Parameters	Test name	STS	RE	Loss	Time
$\lambda = 1, \mu = 1$	A-PINN		1.39e-2	7.57e-4	8.62s
	Partial sharing2		5.03e-4	7.64e-4	13.1s
	Partial sharing2	✓	2.12e-4	3.84e-5	6.33s
$\lambda = -3, \mu = 2$	A-PINN		1.98e-2	8.24e-5	8.63s
	Partial sharing2		1.17e-3	6.66e-5	9.63s
	Partial sharing2	✓	1.72e-4	1.63e-5	4.43s
$\lambda = \frac{9}{2}, \mu = -\frac{5}{2}$	A-PINN		2.34e-2	3.31e-4	8.66s
	Partial sharing2		6.43e-2	4.14e-3	13.8s
	Partial sharing2	✓	3.35e-4	6.92e-6	4.55s

In this subsection, we address the initial-boundary value problem for the following nonlinear partial DIDE:

$$\begin{cases} u_t(x, t) = -u(x, t-1)u_x(x, t) + \int_{t-1}^t e^{s-t}u(x, s)ds + f(x, t), & (x, t) \in \Omega \times [0, 3], \\ u(x, t) = \sin(\pi x), & (x, t) \in \Omega \times [-1, 0], \\ u(0, t) = u(1, t) = 0. \end{cases} \quad (4.3)$$

Here,  $f(x, t)$  is chosen by the exact solution

$$u(t) = \begin{cases} e^t \sin(\pi x), & t \in [0, 1), \\ e^{-\frac{1}{2}t + \frac{3}{2}} \sin(\pi x), & t \in [1, 2), \\ e^{\frac{1}{4}t} \sin(\pi x), & t \in [2, 3]. \end{cases}$$

By introducing an auxiliary output  $v(t)$  to represent the integral in the equation, (4.3)

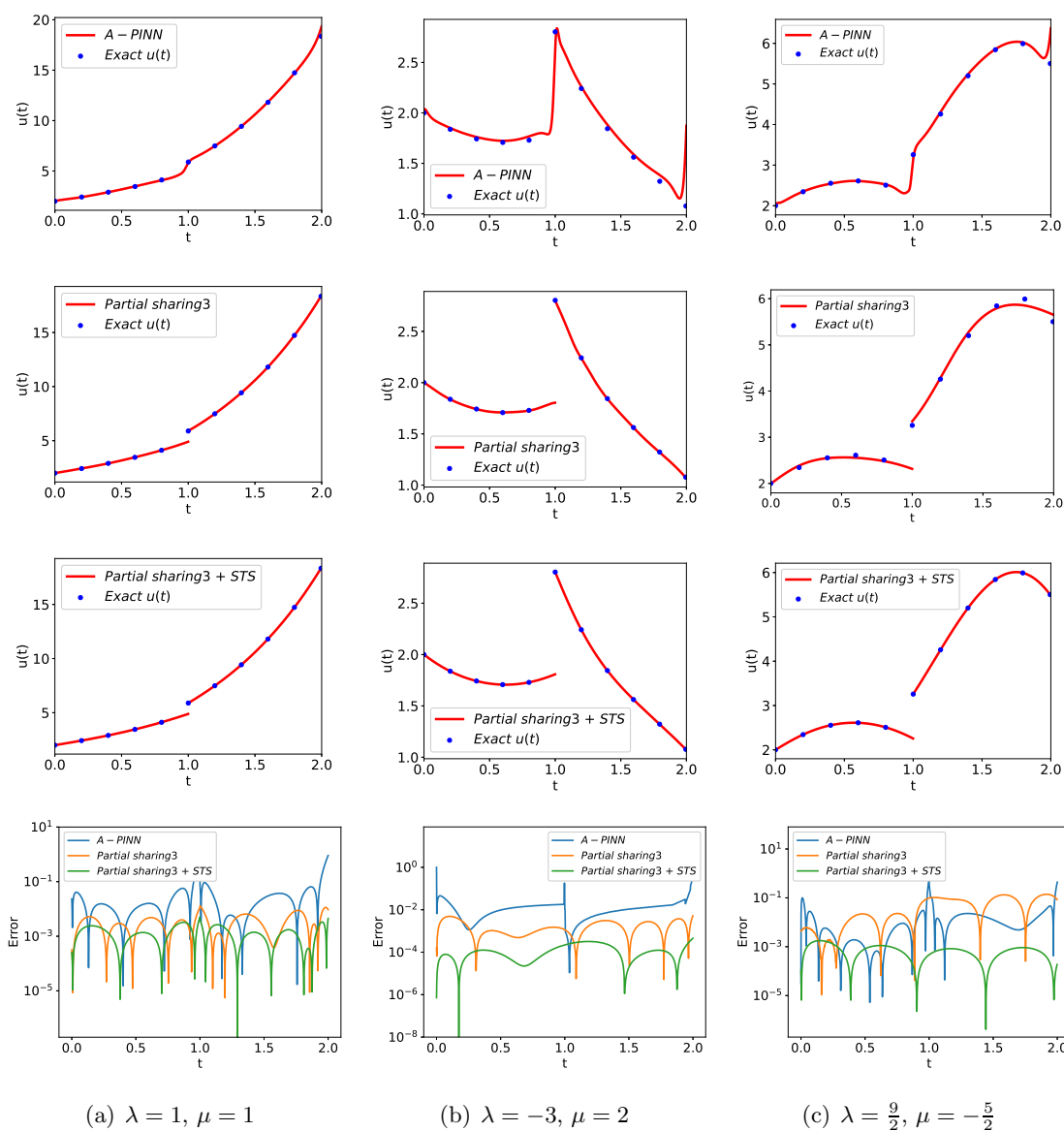


Figure 6 The exact solution, predicted solutions and the absolute value of point-wise errors of diverse schemes for NDIDEs with different parameters.

can be re-expressed as

$$\left\{ \begin{array}{l} u_t(x, t) = -u(x, t-1)u_x(x, t) + v(x, t), \quad (x, t) \in \Omega \times [0, 3], \\ v(x, t) = \int_{t-1}^t e^{s-t} u(x, s) ds, \\ u(0, t) = u(1, t) = 0, \\ u(x, t) = \sin(\pi x), \quad (x, t) \in \Omega \times [-1, 0], \\ v(x, t) = (1 - e^{-1})\sin(\pi x), \quad (x, t) \in \Omega \times [-1, 0]. \end{array} \right.$$

After dividing the time interval  $[0, 3]$  into three subintervals based on the delay, the loss function, which incorporates the boundary condition loss term, can be expressed as:

$$\mathcal{L}(\theta) = \sum_{k=1}^3 \left[ \mathcal{L}_d^k(\theta; \mathcal{N}_d^k) + \mathcal{L}_f^k(\theta; \mathcal{N}_f^k) + \mathcal{L}_{da}^k(\theta; \mathcal{N}_d^k) + \mathcal{L}_a^k(\theta; \mathcal{N}_f^k) + \mathcal{L}_b^k(\theta; \mathcal{N}_b^k) \right],$$

where

$$\mathcal{L}_d^k(\theta; \mathcal{N}_d^k) = \frac{1}{|\mathcal{N}_d^k|} \sum_{(x,t) \in \mathcal{N}_d^k} \left| \hat{u}_k(\hat{\theta}; x, t) - \hat{u}_{k-1}^*(x, t) \right|^2,$$

$$\mathcal{L}_{da}^k(\theta; \mathcal{N}_d^k) = \frac{1}{|\mathcal{N}_d^k|} \sum_{(x,t) \in \mathcal{N}_d^k} \left| \hat{v}_k(\theta; x, t) - \hat{v}_{k-1}^*(x, t) \right|^2$$

are the loss terms of  $u(x, t)$  and  $v(x, t)$  on  $t = k - 1$  by the continuity of  $u$  at the breaking points,

$$\mathcal{L}_f^k(\theta; \mathcal{N}_f^k) = \frac{1}{|\mathcal{N}_f^k|} \sum_{(x,t) \in \mathcal{N}_f^k} \left| \frac{\partial \hat{u}_k(\theta; x, t)}{\partial t} + \hat{u}_{k-1}^*(t-1) \frac{\partial \hat{u}_k(\theta; x, t)}{\partial x} - v_k(\theta, t) \right|^2,$$

$$\mathcal{L}_a^k(\theta; \mathcal{N}_f^k) = \frac{1}{|\mathcal{N}_f^k|} \sum_{(x,t) \in \mathcal{N}_f^k} \left| \frac{\partial \hat{v}_k(\theta; x, t)}{\partial t} - \hat{u}_k(\theta; t) + e^{-1} \hat{u}_{k-1}^*(t-1) + \hat{v}_k(\theta; x, t) \right|^2$$

are the loss terms of the governing equation and the auxiliary output  $v(x, t)$  on the  $k$ -th spatio-temporal domain, and

$$\mathcal{L}_b^k(\hat{\theta}; \mathcal{N}_b^k) = \frac{1}{|\mathcal{N}_b^k|} \sum_{(x,t) \in \mathcal{N}_b^k} \left| \hat{u}_k(\hat{\theta}; x, t) - 0 \right|^2$$

is the loss term for the boundary conditions. Specifically,  $\hat{u}_0^*(t) = \sin(\pi x)$ ,  $\hat{v}_0^*(0) = (1 - e^{-1})\sin(\pi x)$ .

Table 7 presents the five tests related to equation (4.3). Table 8 provides a quantitative comparison of the results, highlighting the optimal outcomes through bolded loss values and RE. It can be observed that no sharing structure exhibits a low initial error, but partial sharing3 ultimately achieves the smallest error.

Table 7 Five testing structures used in subsection 4.3.

Test name	Structure	Layers shared	Layers Independent	Number of Parameters
Partial sharing1	2-30-30/-30-30-2(*3)	2L*30N	2L*30N	6786
Partial sharing2	2-30-30/-39-39-3(*2)	2L*30N	2L*39N	6798
Partial sharing3	2-30-30/-19-19-1(*6)	2L*30N	2L*19N	6954
Full sharing	2-46-46-46-46-6	4L*46N	0L	6906
No sharing	2-23-23-23-23-1(*6)	0L	4L*23N	7182

Next, we employ partial sharing3 to compare with A-PINN. We take  $|\mathcal{N}_i| = 3000$  initial points in  $\Omega \times [-1, 0]$ ,  $|\mathcal{N}_b| = 300$  boundary points at  $x = 0, 1$  and  $|\mathcal{N}_f| = 9300$  collocation points across the entire spatial-temporal domain for A-PINN. The network is structured as 2-46-46-46-46-2, comprising 6718 parameters. For partial sharing3 with or without STS, we take  $|\mathcal{N}_d^m| = 100$  ( $m = 1, 2, 3$ ) breaking points at  $t = 0, 1, 2$ , respectively,  $|\mathcal{N}_b^m| = 100$  ( $m = 1, 2, 3$ ) for each boundary conditions and  $|\mathcal{N}_f^m| = 3000$  ( $m = 1, 2, 3$ ) collocation points for each task. Each of the three schemes is subjected to 1000 iterations, with STS participating in the second and third tasks at the 200th and 500th iterations, respectively.

Figure 7 compares the loss functions during the training of the three schemes used to solve (4.3). The results indicate that the loss function of partial sharing3 with STS is the smallest, presenting a less challenging optimization problem for equation (4.3). Figure 8 presents the comparison of the prediction results of the diverse methods with the true solution, together with the point-wise errors. Both configurations of partial sharing3, with and without STS, yield favorable results, but the error of the former increases more significantly with the increase of  $t$ . Table 9 provides the relative errors and training times for the three schemes. The relative error of partial sharing3 with STS is two orders of magnitude higher than that of A-PINN.

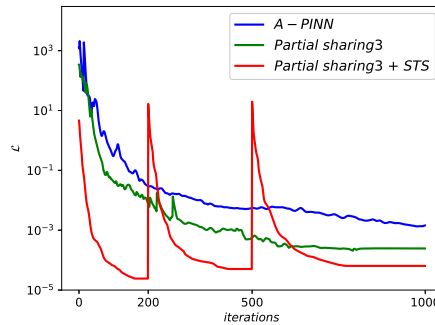


Figure 7 Loss history of diverse schemes for solving equation (4.3).

Table 8 Comparison of testing results among the five structures in Table 7 for solving equation (4.3).

Term	Test name	Iteration-600	Iteration-800	Iteration-1000	Time
Loss	Partial sharing1	<b>3.46e-4</b>	6.64e-5	6.08e-5	47.0s
	Partial sharing2	5.37e-4	7.75e-5	7.46e-5	53.6s
	Partial sharing3	3.87e-4	<b>6.42e-5</b>	<b>6.42e-5</b>	49.9s
	Full sharing	1.20e-3	8.05e-5	6.99e-5	53.3s
	No sharing	4.59e-4	7.50e-5	7.49e-5	<b>40.1s</b>
RE	Partial sharing1	2.64e-3	1.34e-3	1.02e-3	
	Partial sharing2	1.68e-3	1.59e-3	9.73e-3	
	Partial sharing3	1.55e-3	1.12e-3	<b>8.70e-4</b>	
	Full sharing	3.81e-3	1.19e-3	1.25e-3	
	No sharing	<b>1.37e-3</b>	<b>1.07e-3</b>	9.31e-4	

Table 9 Comparison of diverse schemes for solving equation (4.3).

NN structure	STS	RE	Loss	Time
A-PINN		1.36e-2	1.44e-3	65.1s
Partial sharing3		1.51e-3	2.45e-4	84.0s
Partial sharing3	✓	6.69e-4	6.42e-5	49.9s

#### 4.4 The Inverse Problem of Nonlinear DIDE

In this subsection, we consider the inverse problem of the following nonlinear DIDE:

$$\begin{cases} u'(t) = \lambda u(t-1)u(t) + \mu \int_{t-1}^t e^{s-t} u(s) ds, & t \in [0, 3], \\ u(t) = 1, & t \in [-1, 0]. \end{cases} \quad (4.4)$$

We induce an auxiliary output  $v(t)$  to represent the integral, and rewrite (4.4) as

$$\begin{cases} u'(t) = \lambda u(t-1)u(t) + \mu v(t), & t \in [0, 3], \\ v(t) = \int_{t-1}^t e^{s-t} u(s) ds, & t \in [0, 3], \\ u(t) = 1, & t \in [-1, 0], \\ v(t) = e^t - e^{t-1}, & t \in [-1, 0]. \end{cases}$$

After dividing the time interval  $[0, 3]$  into three subintervals based on  $\tau = 1$ , assuming that the reference solutions on the first two subintervals have been obtained, we define the

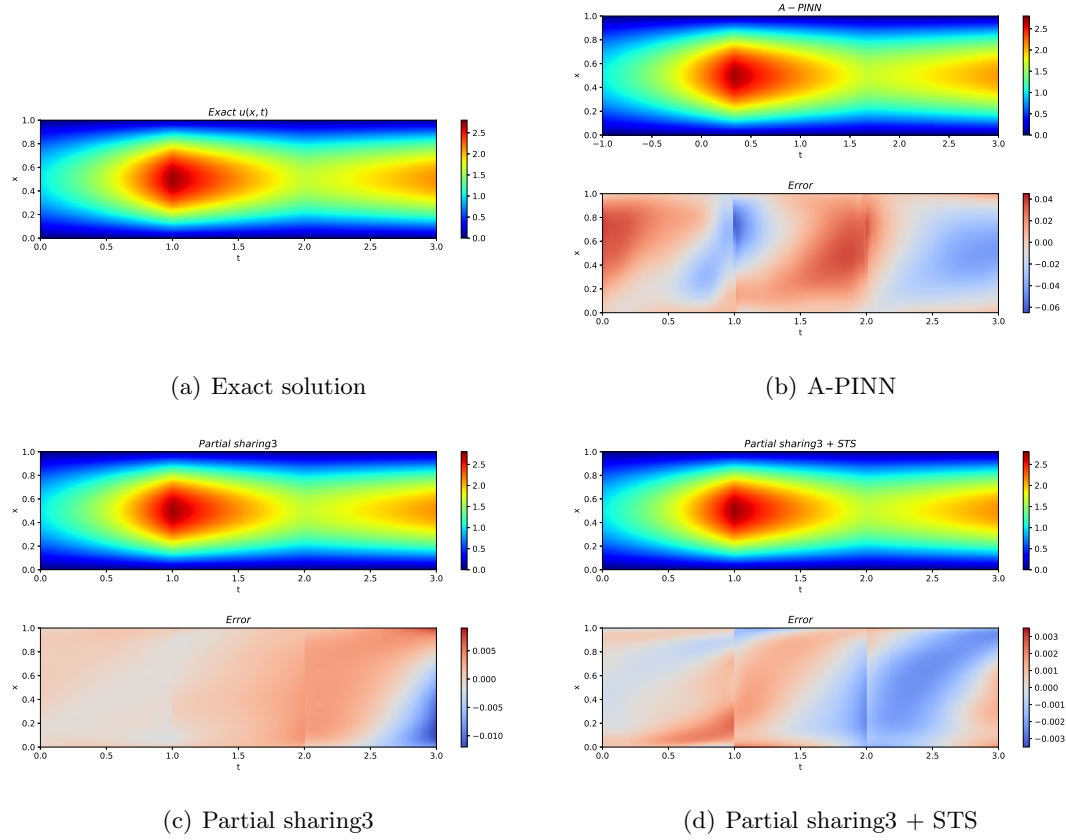


Figure 8 The exact solution, approximate solutions and point-wise errors of different schemes for equation (4.3).

corresponding loss function over all three subintervals as:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}) = & \sum_{k=1}^3 \left[ \mathcal{L}_{d1}^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_d^k) + \mathcal{L}_f^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_f^k) \right. \\ & + \mathcal{L}_{da}^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_d^k) + \mathcal{L}_a^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_f^k) \\ & \left. + \mathcal{L}_{inv}^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_{inv}^k) \right] + \sum_{k=2}^3 \mathcal{L}_{d2}^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_d^k), \end{aligned}$$

where  $\mathcal{L}_{d1}$ ,  $\mathcal{L}_{d2}$  and  $\mathcal{L}_{da}$  are derived from subsection 3.3 under the special case of  $\tau = 1$  and adding trainable parameters.

$$\begin{aligned} \mathcal{L}_f^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_f^k) = & \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \frac{\partial \hat{u}_k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; t)}{\partial t} - \mu v_k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}, t) \right. \\ & \left. - \lambda \hat{u}_{k-1}^*(t-1) \hat{u}_k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; t) \right|^2 \end{aligned}$$

is the loss term of the governing equation on the  $k$ -th subinterval,

$$\mathcal{L}_a^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_f^k) = \frac{1}{|\mathcal{N}_f^k|} \sum_{t \in \mathcal{N}_f^k} \left| \frac{\partial \hat{v}_k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; t)}{\partial t} - \hat{u}_k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; t) + e^{-1} \hat{u}_{k-1}^*(t-1) + \hat{v}_k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; t) \right|^2$$

is the loss term of the auxiliary output  $v(t)$  on the  $k$ -th subinterval and

$$\mathcal{L}_{inv}^k(\boldsymbol{\theta}, \lambda_{inv}, \mu_{inv}; \mathcal{N}_{inv}) = \frac{1}{|\mathcal{N}_{inv}^k|} \sum_{t \in \mathcal{N}_{inv}^k} |\hat{u}_k(\boldsymbol{\theta}, \lambda_{inv}; t) - u_k^{inv}(t)|^2$$

is the loss term of measurement on the  $k$ -th subinterval. Here,  $\mathcal{N}_{inv}^k$  represents the set of measurement points within the  $k$ -th subinterval and  $u_k^{inv}(t)$  denotes the measurement data. Specifically,  $\hat{u}_0^*(t) = 1$ ,  $\hat{v}_0^*(0) = 1 - e^{-1}$ .

Table 10 demonstrates the training results for  $\lambda = -1$  and  $\mu = 1$ . We set the numbers of measurement points  $|\mathcal{N}_{inv}^m| = 50$ ,  $m = 1, 2, 3$  for each subinterval, and the corresponding  $u_m^{inv}(t)$ ,  $m = 1, 2, 3$  are obtained by high-precision finite difference method. Partial sharing2 and partial sharing3 are found to be more effective, we selected the latter for the subsequent phase of the experiment.

Table 10 Comparison of testing results among the five structures in Table 1 for training  $\lambda$  and  $\mu$ .

Term	Test name	Iteration-300	Iteration-400	Iteration-500	Time
$ \lambda - \lambda_{inv} $	Partial sharing1	1.13e-3	1.04e-3	9.21e-4	<b>4.38s</b>
	Partial sharing2	<b>5.48e-4</b>	<b>3.29e-4</b>	<b>3.40e-4</b>	5.69s
	Partial sharing3	2.48e-3	1.22e-3	8.17e-4	5.24s
	Full sharing	2.91e-3	5.75e-4	5.64e-4	5.34s
	No sharing	9.41e-4	8.79e-4	8.41e-4	4.67s
$ \mu - \mu_{inv} $	Partial sharing1	<b>2.12e-3</b>	2.00e-3	1.84e-3	
	Partial sharing2	2.92e-3	<b>8.21e-4</b>	8.74e-4	
	Partial sharing3	2.42e-3	8.27e-4	<b>3.31e-4</b>	
	Full sharing	3.94e-3	1.43e-3	1.40e-3	
	No sharing	2.34e-3	2.26e-3	2.21e-3	

To further analyze the effects of the number of measurement data and the noise level on solving the inverse problem, we employed various amounts of measurement data with distinct noise levels as training data. Noisy data is generated by adding Gaussian noise to the exact values. We tested the number of measurement data ranging from 10 to 500 across each interval, with noise levels varying from 0% to 10%. The absolute value of absolute

errors of the parameters  $\lambda$  and  $\mu$  across different scenarios are presented in Table 11. As the number of measurement data increases, the error of the identified parameters decreases, resulting in more accurate values for  $\lambda$  and  $\mu$ . However, increasing the noise level in the measurement data results in higher errors for the identified parameters. The results indicate that with sufficient measurement data, there is no sensitivity to the noise level, and the unknown parameters can be accurately identified even when the amount of data is limited.

Table 11 The absolute value of absolute errors in the identified  $\lambda$  and  $\mu$  for different numbers of measurement data and noise levels.

Term	$N_{inv}$	0% noise	1% noise	5% noise	10% noise
$ \lambda - \lambda_{inv} $	10	1.70e-3	2.04e-2	5.93e-2	1.89e-1
	30	3.19e-4	2.32e-3	5.48e-3	5.86e-2
	50	8.17e-4	2.58e-3	7.25e-3	3.43e-2
	100	1.96e-3	3.70e-3	7.08e-3	8.51e-3
	200	2.48e-4	1.38e-3	6.56e-3	3.12e-3
	500	2.65e-4	7.14e-4	3.83e-3	3.69e-3
$ \mu - \mu_{inv} $	10	2.35e-3	3.13e-2	7.58e-2	9.51e-2
	30	3.51e-3	3.55e-3	1.19e-2	8.14e-2
	50	3.31e-4	5.43e-3	9.04e-3	1.04e-2
	100	7.76e-4	5.44e-3	2.01e-3	8.95e-3
	200	1.06e-3	1.17e-3	2.04e-2	2.13e-2
	500	4.09e-4	8.63e-4	5.06e-3	7.14e-3

## 5 Conclusions

In this paper, we employed DNNs combining MTL with STS to address both the forward and backward problems of DIDEs. To efficiently incorporate the regularity at breaking points into the loss function, the original problem is divided into multiple tasks based on the delay. We subsequently employed various parameter sharing structures for comparison. To deal with the optimization challenges arising from the increased complexity of the loss function, we adopted STS to generate reference solutions for subsequent tasks. Notably, the additional degrees of freedom offered by the network structure in MTL enable the effective incorporation of properties at the delay points of the equations and the auxiliary outputs into the loss function.

Numerical experiments showed that the DNNs combined with MTL and STS had higher accuracy than A-PINN in solving the DIDEs. Among the parameter sharing structures: no

sharing, full sharing, and partial sharing, the partial sharing structure yields relatively better results, whereas the full sharing structure is less effective than the others. It can be found from the loss values and relative errors that using STS effectively reduced training difficulty and improved approximation accuracy. In solving the inverse problem, we successfully identified the unknown parameters in the equations. Furthermore, this method effectively identifies these parameters even when sparse or noisy measurement data are present.

## References

- [1] Raissi M, Perdikaris P, Karniadakis G E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations[J]. *J. Comput. Phys.*, 2019, 378(1): 686–707.
- [2] Lu Lu, Meng Xuhui, Mao Zhiping, et al. DeepXDE: A deep learning library for solving differential equations[J]. *SIAM Rev.*, 2021, 63(1): 208–228.
- [3] Peng Wei, Zhang Jun, Zhou Weien, et al. IDRLnet: A physics-informed neural network library[J]. arxiv preprint arxiv:2107.04320, 2021.
- [4] Rao Cengping, Sun Hao, Liu Yang. Physics-informed deep learning for computational elastodynamics without labeled data[J]. *J. Eng. Mech.*, 2021, 147(8): 04021043.
- [5] Mao Zhiping, Jagtap A D, Karniadakis G E. Physics-informed neural networks for high-speed flows[J]. *Comput. Meth. Appl. Mech. Eng.*, 2020, 360: 112789.
- [6] Zhu Qiming, Liu Zeliang, Yan Jinhui. Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks[J]. *Comput. Mech.*, 2021, 67: 619–635.
- [7] Yang Liu, Zhang Dongkun, Karniadakis G E. Physics-informed generative adversarial networks for stochastic differential equations[J]. *SIAM J. Sci. Comput.*, 2020, 42(1): A292–A317.
- [8] Karumuri S, Tripathy R, Billionis I, et al. Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks[J]. *J. Comput. Phys.*, 2020, 404: 109120.
- [9] Pang Guofei, Lu Lu, Karniadakis G E. fPINNs: Fractional physics-informed neural networks[J]. *SIAM J. Sci. Comput.*, 2019, 41(4): A2603–A2626.
- [10] Guo Ling, Wu Hao, Yu Xiaochen, et al. Monte Carlo fPINNs: Deep learning method for forward and inverse problems involving high dimensional fractional partial differential equations[J]. *Comput. Meth. Appl. Mech. Eng.*, 2022, 400: 115523.
- [11] Wight C L, Zhao Jia. Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks[J]. arxiv preprint arxiv:2007.04542, 2020.
- [12] Kolmanovskii V, Myshkis A. Applied theory of functional differential equations[M]. Berlin, German: Springer Science & Business Media, 2012.
- [13] Gourley S A, Ruan S. Dynamics of the diffusive Nicholson’s blowflies equation with distributed delay[J]. *P. Roy. Soc. Edinb. A.*, 2000, 130(6): 1275–1291.
- [14] Li Wantong, Ruan Shigui, Wang Zhicheng. On the diffusive Nicholsons blowflies equation with nonlocal delay[J]. *J. Nonlinear Sci.*, 2007, 17: 505–525.

- [15] Xing Baixue, Liu Haixia, Liu Hongliang, et al. Chebyshev neural network method for solving delay-integro-differential-algebraic equations based on variable transformation[J]. PREPRINT (Version 1) available at Research Square, DOI: doi.org/10.21203/rs.3.rs-3173404/v1, 2023.
- [16] Yuan Lei, Ni Yiqing, Deng Xiangyun, et al. A-PINN: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations[J]. J. Comput. Phys., 2022, 462: 111260.
- [17] Zhang Yu, Yang Qiang. A survey on multi-task learning[J]. IEEE Trans. Knowl. Data Eng., 2021, 34(12): 5586–5609.
- [18] Goodfellow I. Deep learning[M]. Cambridge, MA: MIT Press, 2016.
- [19] Diao Yu, Yang Jianchuan, Zhang Ying, et al. Solving multi-material problems in solid mechanics using physics-informed neural networks based on domain decomposition technology[J]. Comput. Meth. Appl. Mech. Eng., 2023, 413: 116120.
- [20] Bischof R, Kraus M. Multi-objective loss balancing for physics-informed deep learning[J]. arxiv preprint arxiv:2110.09813, 2021.
- [21] Long Mingsheng, Cao Zhangjie, Wang Jianmin, et al. Learning multiple tasks with multilinear relationship networks[C]// Adv. Neural Inf. Process. Syst., New York, NY: Curran Associates, Inc., 2017, 30.
- [22] Baydin A G, Pearlmutter B A, Radul A A, et al. Automatic differentiation in machine learning: a survey[J]. J. Mach. Learn. Res., 2018, 153(18): 1–43.
- [23] Brunner H. Collocation methods for Volterra integral and related functional differential equations[M]. Cambridge: Cambridge university press, 2004.
- [24] Brunner H, Zhang Wenkui. Primary discontinuities in solutions for delay integro-differential equations[J]. Methods Appl. Anal., 1999, 6(4): 525–534.
- [25] Stein M. Large sample properties of simulations using Latin hypercube sampling[J]. Technometrics, 1987, 29(2): 143–151.

## 基于结合多任务学习的深度神经网络求解延迟积分微分方程

王辰尧, 史 峰

(哈尔滨工业大学(深圳)理学院, 广东 深圳 518055)

**摘要:** 深度神经网络 (DNNs) 在求解非线性偏微分方程 (PDEs) 的正问题反问题方面都是有效的。然而, 传统DNN方法在处理具有常延迟的延迟微分方程 (DDEs) 和延迟积分微分方程 (DIDE) 等问题时往往遇到困难, 主要因为这些方程在由延迟产生的断点处的正则性较低。本文提出了一种结合多任务学习 (MTL) 的DNN方法来求解DIDE的正问题和反问题。这种方法的核心思想是根据延迟将原始方程划分为多个任务, 使用辅助输出表示积分项, 然后结合MTL将断点处的性质无缝纳入到损失函数中。此外, 鉴于多个任务和输出引起的训练难度增加, 我们采用顺序训练方案来降低训练复杂性, 并为后续任务提供参考解。与传统DNN方法相比, 这种方法显著提高了使用DNN方法求解DIDE的精度。我们通过几个数值算例验证了该方法的有效性, 测试了MTL中的多种参数共享结构, 并比较了这些结构的效果。最后, 将该方法应用于求解非线性DIDE的反问题, 结果表明, 此方法可以从稀疏或有噪声的数据中发现DIDE的未知参数。

**关键词:** 延迟积分微分方程; 多任务学习; 参数共享结构; 深度神经网络; 顺序训练方案

MR(2010)主题分类号: 45J05; 45K05 中图分类号: O242.2