A piecewise neural network method for solving large interval solution to initial value problem of ordinary differential equations

Dongpeng Han[1] and Chaolu Temuer[2*†]

[1,2]Mathematics Department, Shanghai Maritime University, 1550 Haigang Avenue, Shanghai, 201306, Shanghai, China.

*Corresponding author(s). E-mail(s): tmchaolu@shmtu.edu.cn;
Contributing authors: handongpeng@stu.shmtu.edu.cn;
[†]These authors contributed equally to this work.

Abstract

Various traditional numerical methods for solving initial value problems of differential equations often produce local solutions near the initial value point, despite the problems having larger interval solutions. Even current popular neural network algorithms or deep learning methods cannot guarantee yielding large interval solutions for these problems. In this paper, we propose a piecewise neural network approach to obtain a large interval numerical solution for initial value problems of differential equations. In this method, we first divide the solution interval, on which the initial problem is to be solved, into several smaller intervals. Neural networks with a unified structure are then employed on each sub-interval to solve the related sub-problems. By assembling these neural network solutions, a piecewise expression of the large interval solution to the problem is constructed, referred to as the piecewise neural network solution. The continuous differentiability of the solution over the entire interval, except for finite points, is proven through theoretical analysis and employing a parameter transfer technique. Additionally, a parameter transfer and multiple rounds of pre-training technique are utilized to enhance the accuracy of the approximation solution. Compared with existing neural network algorithms, this method does not increase the network size and training data scale for training the network on each sub-domain. Finally, several numerical experiments are presented to demonstrate the efficiency of the proposed algorithm.

1

1 Introduction

As we know, constrained by convergence requirements, various traditional numerical solution methods, such as Picard approximation and Runge-Kutta methods, for initial value problems (IVP) of differential equations or dynamic systems (ODEs), are typically suitable only for local solutions within a neighborhood interval of the initial value point, even if the problem possesses larger interval (global) solutions. Consequently, extending a local solution to a larger interval remains a fundamental challenge in the field of numerical solutions to ODE problems, which has yet to be fully addressed [1, 2].

Currently, the artificial neural network (ANN) algorithm, as a novel numerical solution method for solving differential equation problems, is rapidly evolving and attracting attention and exploration from many researchers [3]. Although the ANN algorithm, like traditional methods, performs well mainly in the neighborhood of the initial value point and lacks inherent learning capability on large intervals, it offers numerous advantages in solving IVPs of ODEs. For instance, the neural network-based solution of a differential equation is differentiable and presented in closed analytic form, suitable for subsequent calculations. It is independent of discrete schemes and the shape of the variable domain. Therefore, it is anticipated that the aforementioned solution extension problem can be effectively addressed through the enhancement and efficient utilization of the ANN algorithm. In this paper, we propose a piecewise ANN (PWNN) method to tackle this problem.

Before describing our algorithm, let us briefly review the development profile of ANN algorithms for solving differential equation problems.

The theoretical foundation supporting the use of ANN algorithms for solving differential equation problems lies in the general approximation theorems provided by Hornik, Womik, and Li X et al. [4–6], which theoretically establish that any continuous function can be approximated by a neural network. Subsequently, a widely utilized ANN algorithm for solving differential equations with initial boundary value conditions was introduced by Lagaris et al. [7]. However, for problems involving more complex initial boundary values, this method faces difficulties in constructing the necessary trial solutions for the underlying problems.

Another significant development in ANN algorithms for solving differential equation problems is the implementation of automatic differentiation technology [8–11], which has led to the proliferation of ANN methods for solving differential equation problems and their broader application. Building upon this technique, Justin Sirignano and Konstantinos Spiliopoulos proposed the Deep Galerkin method (DGM), a deep neural network method for solving (partial) differential equation problems in higher dimensions [12]. Cosmin Anitescu et al. introduced an ANN method utilizing adaptive configuration strategies to enhance method robustness and reduce computational costs in solving boundary value problems of differential equations [13].

A landmark advancement in ANN algorithms for solving differential equation problems was the introduction of physical information neural networks (PINN) by Raissi et al. [14]. In this method, the initial boundary conditions of the differential equation are incorporated into the loss function, allowing the ANN to directly express an approximate solution to the differential equation. This eliminates the need to construct a trial solution according to equations and initial boundary value conditions, as required

by Lagaris's method. Subsequently, numerous ANN algorithms based on PINN have been developed rapidly. For example, Lei Yuan and Yi-Qing Ni et al. introduced the Auxiliary PINN (A-PINN), capable of bypassing limitations in integral discretization and solving forward and inverse problems of nonlinear integral differential equations [15]. Pao-Hsiung Chiu et al. proposed novel PINN methods for coupling neighboring support points and their derivative terms obtained by automatic differentiation [16]. Yao Huang et al. combined PINN with the homotopy continuation method, proposing Homotopy PINN (HomPINN) for solving multiple solutions of nonlinear elliptic differential equations, overcoming the limitation of PINN in finding only the flattest solution in most cases [17]. Fang, Yin et al. utilized PINN to address a range of femtosecond optical soliton solutions pertaining to the high-order nonlinear Schrödinger equation [18]. Based on the PINN method, Bai, Yuexing et al. proposed an enhanced version of PINN called IPINN, introducing localized adaptive activation functions to improve performance, successfully applying the method to solve several differential equation models in finance [19]. Meng et al. introduced a modified PINN method called PPINN, dividing a long-period problem into a series of short-period ones to accelerate the training of ANN algorithms [20].

Moreover, there are numerous other analogues of PINN-based studies and various types of ANN methods. For instance, convolutional ANN methods [21–23] and theoretically guided ANN methods [24, 25] have also been employed to explore new solution methods for (partial) differential equations. Particularly noteworthy is the work of Run-Fa Zhang and Sudao Bilige, who proposed bilinear neural networks, the first attempt to obtain analytical solutions to nonlinear partial differential equations using the ANN method [26, 27].

Importantly, a general observation from these literature is that ANN algorithms for solving IVPs of differential equations perform well in relatively small domains near the initial value point but sometimes exhibit poor convergence over large intervals. To obtain a large interval solution, it is often necessary to enhance the training data and expand the network size (depth and width). However, doing so not only reduces computational efficiency but also makes it challenging to guarantee obtaining a large interval solution to the problem. Therefore, alternative strategies are being explored to enhance the efficiency of ANN algorithms. For example, various adaptive activation functions [19, 28–30] and adaptive weights [31, 32] have been proposed. The key feature of these studies is the introduction of hyperparameters into traditional activation functions to adjust ANN convergence. Studies by Jaftap et al. have shown that these methods are primarily effective in the early stages of network training [33]. Adaptive weights leverage gradient statistics to optimize the interplay between various components in the loss function by incorporating additional weighting during the training process. While these solutions alleviate the convergence difficulties of neural networks in complex problems from different perspectives, there is insufficient evidence to demonstrate that simultaneously using different optimization schemes can still reduce network training difficulty and achieve large interval solutions. To solve the IVP problem of dynamic systems, Wen et al. proposed an ANN algorithm based on the Lie symmetry of differential equations [34–37]. This novel approach combines Lie group theory and neural network methods.

Given the aforementioned challenges, this paper proposes an innovative approach for obtaining large interval approximate solutions to IVPs of ODEs. In this algorithm, the interval is divided into several small compartments, and a neural network solution is learned on each compartment using PINN. Consequently, the trained ANNs generate fragments of the large interval solution of the IVP of an ODE on those sub-intervals, respectively. For each specific sub-interval, the ANN training requires neither complex structure nor dependence on large-scale training data, significantly reducing computational overhead. Subsequently, by assembling these neural network solutions, a piecewise expression of the large interval solution to the problem is constructed, termed the piecewise neural network solution. The compatibility of these sub-interval network solutions and the continuity and differentiability of the piecewise solution over the entire interval are investigated theoretically. Transfer learning techniques of network parameters between adjacent neural networks and multiple rounds of pre-training approaches in the training procedure are utilized.

The innovative contributions of this paper are threefold. First, a new ANN method for solving the extension problem of the local solution of the initial value problem of differential equations is presented. Second, a novel method of generalizing ANN and its application is introduced. Third, a new approach is explored in which modern algorithms (artificial neural networks, deep learning) are used to overcome the shortcomings of traditional methods for solving numerical solutions to differential equation problems. Our work aims to provide a more comprehensive understanding of the ANN method to solve ODE problems in both theoretical and practical aspects.

The structure of this paper is organized as follows. For completeness, Section 2 briefly reviews the PINN method for IVPs of ODE systems; Section 3 details the PWNN algorithm proposed in this paper; Section 4 presents the theoretical analysis of PWNN solutions and implementation of the proposed method; Section 5 demonstrates the applications of the proposed method in solving several specific IVPs of ODEs. Concurrently, comparisons between the results of our method and those of the PINN method and Runge-Kutta method are presented to demonstrate the efficiency of the proposed algorithm. Finally, Section 6 summarizes the work.

2 A brief recall of PINN

Consider an IVP of a system of differential equations

$$
\begin{cases}
\dfrac{\mathrm{d}y}{\mathrm{d}x} = f(x,y), x \in I = (0,T), \\
y(0) = y^0,
\end{cases}
\tag{1}
$$

with independent variable $x \in R$ and dependent variable $y = (y_1, y_2, \cdots, y_n) \in R^n$ and initial value (or initial condition, IC) $y^0 = (y_1^0, y_2^0, \cdots, y_n^0) \in R^n$. We assume through the article that the right hand side function $f(x,y) = (f_1(x,y), f_2(x,y), \cdots, f_n(x,y)$ satisfies the condition [1]

$(\mathbf{H}_1)$: Continuous in $(x,y) \in \Omega \subset R \times R^n$ for a domain Ω and Lipschitz continuous on y.

The condition guarantees the existence of unique local solution to the problem (1).

4

In addition, since the present article works on the large interval solution of IVP (1), we also use the default assumption below:

(H$_2$): The solution of IVP (1) exists on large interval I of x (called large interval solution).

Thus, the PINN is an ANN algorithm for solving the solution of problem (1) with fully connected structure that consists of one-dimensional input layer (0the layer), M hidden layers, s_i nonlinear active neurons for ith layer with $i = 0, 1, \cdots, M + 1$ and $s_0 = 1, s_{M+1} = n$, and output layer ($(M + 1)$th layer) with n linear output neurons. The output of the network is denoted as

$$N(x) = (N_1(x), N_2(x), \cdots, N_n(x)),$$

for an input sample $x \in I$ which has mathematically the form of

$$N(x) = \sigma_M(\sigma_{M-1}(\cdots(\sigma_1(W^1 t + b^1)W^2 + b^2)\cdots)W^M + b^M)W^{M+1} + b^{M+1}, \quad (2)$$

for selected activation functions σ_{i+1}, weights matrix W^{i+1} between ith and $(i+1)$th layer and bias vector b^{i+1} for $(i + 1)$th layer neurons with $i = 0, 1, 2, \ldots, M$ and $b_{M+1}(x) = x$.

The collection of all trainable parameters of the ANN, denoted as ϑ, is consisted of weights $W^i = (w^{i,1}, w^{i,2}, \ldots, w^{i,s_i})^T$ with $w^{i,j} = (w_i^{1,j}, w_i,^{2,j}, \cdots, w_i^{s_{i-1},j})$ in which the components $w_i^{k,j}$ $(k = 1, 2, \cdots, s_{i-1}, j = 1, 2, \cdots, s_i)$ are the weights connected the neurons in $(i - 1)$th layer to jth neuron in ith layer, and layer biases $b^i = (b_i^1, b_i^2, \ldots, b_i^{s_i})^T (i = 1, 2, \cdots, M+1)$. Hence, W^i is a $s_i \times s_{i-1}$ matrix. Let ϑ denote the set of all weights and biases parameters of the net work. The unsupervised learning (training) method is used. That is, the optimal solution $N(x)$ at optimal parameters $\vartheta = \vartheta^*$ is searched such that the loss function to be given below is minimized.

Usually, the mean square error (MSE) is used to define the loss function for training the network as follows:

$$Loss = \frac{1}{n}\sum_{i=1}^{n} MSE_{f_i} + \frac{1}{n}\sum_{i=1}^{n}(N_i(0) - y_i^0)^2, \quad (3)$$

where

$$MSE_{f_i} = \frac{1}{M}\sum_{j=1}^{M}(N_i'(x_j) - f_i(x_j, N_1(x_j), N_2(x_j), \cdots, N_n(x_j)))^2,$$

and $\{x_j\}_{j=1}^{M}$ is a training data set sampled by a distribution sense from the solution interval I, and $N_i'(x)$ is the derivative of the approximate solution $N_i(x)$ with respect to the independent variable x. In training the network, the automatic differentiation operation ∇ is used to realize $N_i'(x)$. One of initialization methods [38] is performed on the neural network parameters when starting the training. A optimization technique [39] is used to adjust the network parameters in order to minimize the loss function (3) and carry out the back propagation of errors. After completing the training of the

network by minimizing the loss function ($Loss < \epsilon$) or sufficient iterative parameter refinement (learning, training) ($Iter > maxit$), we obtain the network pattern (2) with a optimal parameter set ϑ^*, which represents the approximate solution of problem (1). The basic structure of above mentioned ANN (e.g, PINN) is shown in Fig. (1).

Particularly, what should be especially mentioned here that the experiences of applying an ANN on solving IVP of a (partial) differential equations show that it often yields high accuracy approximate solution merely near the initial value point rather than the entire interval $[0, T)$ [1]. That is, generally, there exist a constant $0 < \delta \ll T$ such that the output $N(x)$, called PINN solution, converges to exact solution $y(x)$ over $[0, \delta)$ with high accuracy. We denote the approximation as

$$N(x) \approx y(x).$$

In other words, as the training point gradually moves away from the starting point $x = 0$, the learning ability or generalization ability of the ANN gradually weakens. There are two main potential reasons for this phenomenon. First, the requirement of $Loss \to 0$ in network training is only a necessary condition for $N(x)$ to solve IVP (1). Second, with the expansion of the sampling data interval, the support of initial value information to the training ANN may gradually decline [7]. Therefore, in order to
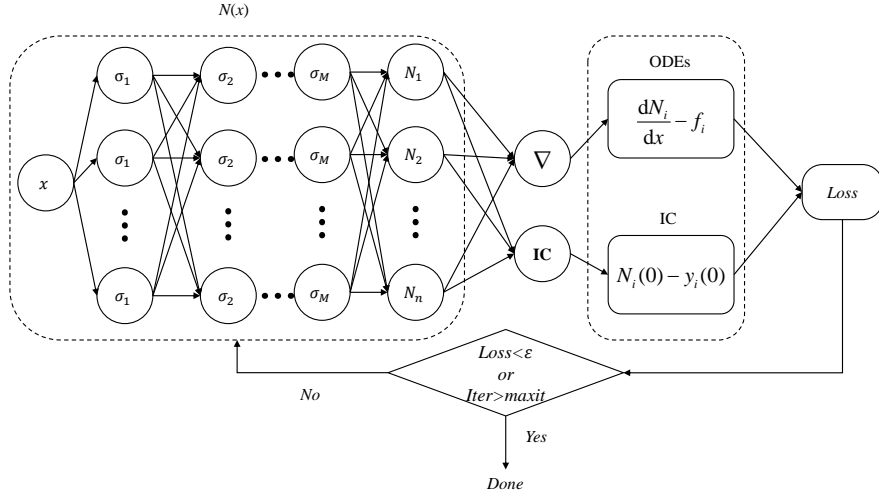

Fig. 1 Network structure of PINN for solving IVP (1).

obtain the large interval solution of problem (1) by ANN method, it is often necessary to construct a neural network with a large number of neurons and a deep layers so that the influence of the initial value is deep and wide, which may make the network learn the solution as accuracy as possible [3]. However, the increase of network scale significantly increases the training difficulty of the neural network, and even makes it impossible to get normal training. In view of this problem, based on the advantage of PINN's high-precision learning ability in a neighborhood of initial value point, we

propose a PWNN method in next section to obtain the large interval solution. In the method, the solution interval I is split into several intercell compartments and a related small size in scale neural network is applied on each the sub-interval for solving a sub-IVP corresponding to (1). Finally stitches these neural network solutions of the sub-intervals together to obtain the approximate solution of problem (1) over whole interval I.

3 A piecewise ANN method

In this section, based on the default assumptions $(H_1)-(H_2)$ and well learning ability of a PINN, we will present the PWNN method for finding the large interval approximate solution of IVP (1).

3.1 Method

The basic framework of the method is described as follows.

First, we divide the interval $[0, T]$ into p sub-intervals, namely insert $p - 1$ points in interval $[0, T]$ making

$$0 = a_0 < a_1 < \cdots < a_p = T, \ [0, T] = \bigcup_{i=1}^{p} \Delta_i, \ \Delta_i = [a_{i-1}, a_i],$$

and suppose that sub-interval Δ_i is relative small so that the well performing of PINN is guaranteed. Then, we construct p ANNs (e.g. PINNs) with identical structures as given in Section 2 corresponding to the splitting of the interval $[0, T]$, denoted them as $\{m_i\}_{i=1}^p$. Each neural network m_i is used to solve a sub-IVP of the same differential equations in (1) on the ith sub-interval Δ_i with a well defined initial value at a_{i-1}. The output of the network is denoted as $N^i(x) = (N_1^i(x), N_2^i(x), \cdots, N_n^i(x))$, which has composite expression like (2) and represents the approximation of exact solution of the sub-IVP, denoted as $\bar{y}^i(x) = (\bar{y}_1^i(x), \bar{y}_2^i(x), \cdots, \bar{y}_n^i(x))$. Particularly, component $N_j^i(x)$ is the output of j-th neuron of network m_i in the output layer for an input $x \in \Delta_i$, representing the approximation of $\bar{y}_j^i(x)$ for $j \in \{1, 2, \cdots, n\}$. The network is illustrated in Fig. 2.

Specifically, first, a PINN m_1 is trained to solve the sub-IVP of the system in (1) on interval Δ_1 with initial value $\bar{y}^1(0) = y^0$. The loss function used for the network is defined as

$$Loss_1 = \frac{1}{M_1 n} \sum_{i=1}^{n} \sum_{j=1}^{M_1} (\frac{dN_i^1}{dx}(x_j) - f_i(x_j, N_1{}^1(x_j), N_2{}^1(x_j), \cdots, N_n^1(x_j)))^2$$

$$+ \frac{1}{n} \sum_{i=1}^{n} (N_i^1(0) - y_i^0)^2.$$

Here, the training data points $\{x_j\}_{j=1}^{M_1}$ are taken from the interval Δ_1 by some distribution sense (the following-up data samplings are similar). The training of the
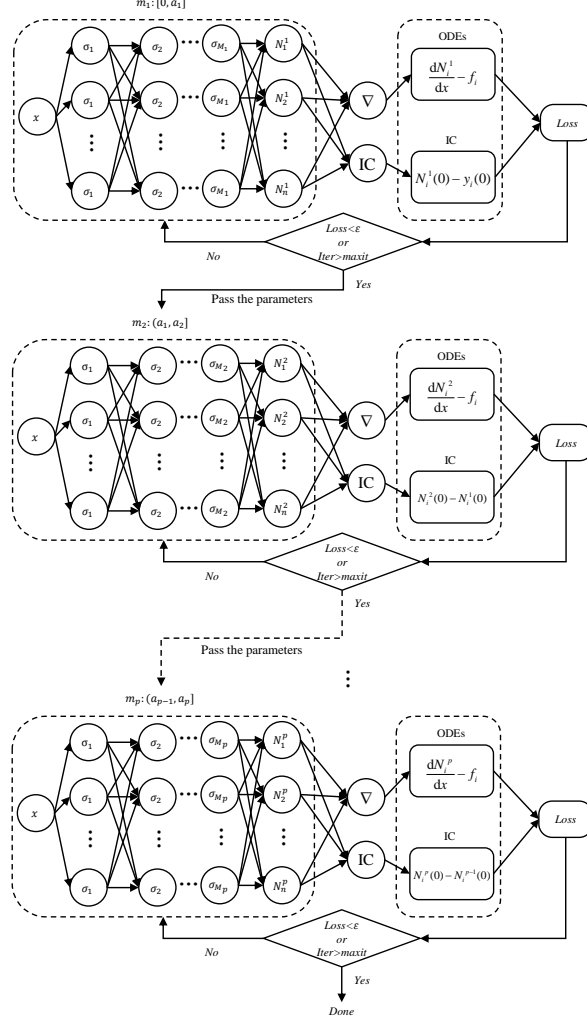
Fig. 2 Network structure of PWNN for solving IVP (1).

network m_1 is completed when the termination (convergence) conditions of the network training (generally the loss function value drops to near zero or the iteration step size is sufficient) are satisfied. As a result, due to the approximation property of ANN, the ANN is an approximate solution of the sub-IVP, denoted by $N^1(x) = (N_1^1(x), N_2^1(x), \cdots, N_n^1(x))$, i.e., it is the approximation of exact solution $\bar{y}^1(x) = (\bar{y}_1^1(x), \bar{y}_2^1(x), \cdots, \bar{y}_n^1(x))$ of the sub-IVP over Δ_1. Notably, compared to a neural network that solves IVP (1) over the entire interval I, network m_1 requires only a structure of relatively small scale and is trained well on Δ_1 because it is a small neighborhood of the initial value point $x = 0$. This gives full play to the strong local learning ability, faster convergence speed and high approximation accuracy of PINN.

After obtaining the first network m_1 on Δ_1, we train and use the next network m_2 to approximately solve the sub-IVP of the system in (1) with an initial value of $\bar{y}^2(a_1) = N^1(a_1)$ in terms of previous trained network m_1 on the interval Δ_2. The corresponding loss function for m_2 is defined as follows:

$$
\begin{aligned}
Loss_2 = & \frac{1}{M_2 n} \sum_{i=1}^{n} \sum_{j=1}^{M_2} \left(\frac{\mathrm{d}N_i^2}{\mathrm{d}x}(x_j) - f_i(x_j, N_1{}^2(x_j), N_2{}^2(x_j), \cdots, N_n^2(x_j)) \right)^2 \\
& + \frac{1}{n} \sum_{i=1}^{n} \left(N_i^2(a_1) - N_i^1(a_1) \right)^2 .
\end{aligned}
\tag{4}
$$

Here, the training data points $\{x_j\}_{j=1}^{M_2}$ are taken from the interval Δ_2. The second term on the right-hand side of the loss function ensures that network m_2 approximates m_1 as closely as possible at the initial point a_1. When the termination conditions are satisfied network m_2 has been trained and yields approximate solution $N^2(x) = (N_1^2(x), N_2^2(x), \cdots, N_n^2(x))$ of the exact solution $\bar{y}^2(x) = (\bar{y}_1^2(x), \bar{y}_2^2(x), \cdots, \bar{y}_n^2(x))$ of the sub-IVP.

Inductively, suppose we have already trained network m_i and obtained approximate solution $N^i(x) = (N_1^i(x), N_2^i(x), \cdots, N_n^i(x))$ of the exact solution $\bar{y}^i(x) = (\bar{y}_1^i(x), \bar{y}_2^i(x), \cdots, \bar{y}_n^i(x))$ of the sub-IVP of system in (1) with initial value $\bar{y}^i(a_i) = N^{i-1}(a_{i-1})$ on the interval Δ_i for $i = 1, 2, \cdots, k-1$. Now, next network m_k is trained and used to approximate exact solution $\bar{y}^k(x)$ of the sub-IVP of the same system in (1) on the interval Δ_k with initial value $\bar{y}^k(a_{k-1}) = N^{k-1}(a_{k-1})$ in terms of previous network m_{k-1}. The training data points $\{x_j\}_{j=1}^{M_k}$ are taken from the interval Δ_k. The corresponding loss function for network m_k is defined as:

$$
\begin{aligned}
Loss_k = & \frac{1}{M_k n} \sum_{i=1}^{n} \sum_{j=1}^{M_k} \left(\frac{\mathrm{d}N_i^k}{\mathrm{d}x}(x_j) - f_i(x_j, N_1{}^k(x_j), N_2{}^k(x_j), \cdots, N_n^k(x_j)) \right)^2 \\
& + \frac{1}{n} \sum_{i=1}^{n} \left(N_i^k(a_{k-1}) - N_i^{k-1}(a_{k-1}) \right)^2 .
\end{aligned}
\tag{5}
$$

When the termination conditions are satisfied network m_k has been trained and yields approximate solution $N^k(x) = (N_1^k(x), N_2^k(x), \cdots, N_n^k(x))$ of the exact solution $\bar{y}^k(x) = (\bar{y}_1^k(x), \bar{y}_2^k(x), \cdots, \bar{y}_n^k(x))$ of the sub-IVP.

In this way we train and use all ANNs m_i or equivalently $N^i(x)$, subsequently to approximate exact solution $\bar{y}^i(x)$ of the sub-IVP over sub-intervals Δ_i with initial value $\bar{y}^i(a_{i-1}) = N^{i-1}(a_{i-1})$ for $i = 1, 2, \cdots, p$.

Finally, we construct a piecewise function over $[0, T)$ by these PINN's solutions given by:

$$\hat{y}(x) = \begin{cases} N^1(x), & x \in [0, \ a_1], \\ N^2(x), & x \in (a_1, \ a_2], \\ \vdots \\ N^p(x), & x \in (a_{p-1}, T), \end{cases}$$

where

$$N^k(x) = (N_1^k(x), N_2^k(x), \cdots, N_n^k(x)), \ k = 1, 2, \cdots, p.$$

4 Theoretical analysis and a parameter transfer method

In the section, we will theoretically demonstrate that the function $\hat{y}(x)$ given in last section is an approximation of the large interval solution $y(x)$ of IVP (1) we are looking for. Meantime, we give a parameter transfer method and multiple rounds of pre-training approach in training PWNN.

4.1 Approximation of large interval solution

In fact, the network m_k, or equivalently $N^k(x)$ with $N^0(x) = y^0$, presented in the last section approximately solved the following sub-IVP of the system in (1):

$$\begin{cases} \dfrac{\mathrm{d}\bar{y}^k}{\mathrm{d}x} = f(x, \bar{y}^k), \ x \in \Delta_k, \\ \bar{y}^k(a_{k-1}) = N^{k-1}(a_{k-1}), \end{cases} \tag{6}$$

for dependent variable $\bar{y}^k(x) = (\bar{y}_1^k(x), \bar{y}_2^k(x), \cdots, \bar{y}_n^k(x))$ and for each $k \in \{1, 2, \ldots, p\}$, inductively, and we have

$$N^k(x) \approx \bar{y}^k(x) \ \ \text{on} \ \ \Delta_k, \ k = 1, 2, \cdots, p. \tag{7}$$

In addition, we also set a sub-IVP of (1) on each sub-interval Δ_k as

$$\begin{cases} \dfrac{\mathrm{d}y^k}{\mathrm{d}x} = f(x, y^k), x \in \Delta_k, \\ y^k(a_{k-1}) = y^{k-1}(a_{k-1}), \end{cases} \tag{8}$$

with $y^k(x) = (y_1^k(x), y_2^k(x), \cdots, y_n^k(x))$ and $y^0(0) = y^0$ for $k \in \{1, 2, \cdots, p\}$. Due to the uniqueness of the solution of the IVP of an ODEs under assumption (H$_1$) given in last section, we know that solution $y^k(x)$ of sub-IVP (8) is the restriction of the exact solution $y(x)$ of IVP (1) on Δ_k. That is,

$$y^k(x) = y(x) \ \ \text{on} \ \ \Delta_k, \ k = 1, 2, \cdots, p, \tag{9}$$

10

and particularly, $y(x) = y^1(x) = \bar{y}^1(x)$ on Δ_1. Hence, noticing (7), approximation

$$N^1(x) \approx y(x), \tag{10}$$

holds on Δ_1.

For $k = 2$, we have

$$y^2(a_1) \overset{(8)}{=} y^1(a_1) = \bar{y}^1(a_1) \overset{(7)}{\approx} N^1(a_1) \overset{(6)}{=} \bar{y}^2(a_1).$$

Hence, on sub-interval Δ_2 the initial values of IVP (6) and IVP (8) are approximate. Consequently, by the qualitative property that the solution of IVP of an ODEs with assumption (H_1) continuously depends on the initial value, approximation

$$N^2(x) \approx y(x), \tag{11}$$

holds on Δ_2.

Similarly, we can inductively prove

$$N^k(x) \approx y(x), \tag{12}$$

hold on intervals Δ_k for $k = 1, 2, \cdots, p$. This shows that approximation $\hat{y}(x) \approx y(x)$ holds on each Δ_k.

Furthermore, from the construction of PINN $N(x)$ (see the multiple composite structure of the PINN's output expression (2)), we see that function $N^k(x)$ is continuous and differential over the each sub-interval Δ_k. Therefore, except the end points a_k of $\Delta_k(k = 1, 2, \cdots, p-1)$, function $\hat{y}(x)$ is continuous and differential over large interval $[0, T]$. Although we do not confirm the continuity of $\hat{y}(x)$ on thoses end points, we, from (6) and (7), know approximation

$$N^k(a_k) \approx N^{k+1}(a_k),$$

implying the approximation of the two side limitations

$$\hat{y}(a_k^-) \approx \hat{y}(a_k^+), \tag{13}$$

hold for $k = 1, 2, \cdots, p-1$. This indicates that there may exist just a small 'jump' in the value of function $\hat{y}(x)$ at an end point. However, in numerical solution, the errors can be controlled within the tolerable error range by improving the convergence accuracy of PINN solutions $N^k(x)$.

To sum up above procedure, we have proved the following conclusions.

Theorem: On the solution interval $[0, T]$, the piecewise function $\hat{y}(x)$ satisfies the approximation $\hat{y}(x) \approx y(x)$ and is continuously differentiable except for the finite points $a_k(k = 1, 2, \cdots, p-1)$, where the approximation (13) holds.

The Theorem is the theoretical basis of our PWNN method.

4.2 Transfer of network parameters

According to the research by Xavier et al., the parameter initialization of a neural network directly affect whether the network can be trained successfully [38]. In the proposed PWNN method, we need to train p interrelated PINNs. Therefore, the parameter initialization and transfer of the PWNNs in training them are a key step to successfully obtain large interval approximate solution $\hat{y}(x)$. In order to assess and improve the stability and performance in certain cases, there are some additional network parameter transfer and multiple rounds of pre-training techniques which we employ beyond the basic setup.

1. A parameter transfer technique. In fact, we have already used a parameter initialization technique in the training and theoretical analysis of PWNN. This is shown in IVP (6) and procedure of constructing ANNs $m_k(k = 1, 2, \cdots, p)$, which is explained in more details in the following steps.

Start with $k = 1$. By the standard steps of PINN, initializing network parameters ϑ_1 with Xavier technique or some other ones, we then train first PINN m_1 on Δ_1 to solve IVP (6) for $k = 1$ and determine $N^1(x)$.

For $k = 2, 3, \cdots, p$, initializing parameters ϑ_k of m_k using parameters ϑ_{k-1} obtained in m_{k-1}, we then train the kth PINN m_k on Δ_k to solve IVP (6) and determine $N^k(x)$. That is, once network m_{k-1} has completed training, we pass parameters ϑ_{k-1} to network m_k as initialization of parameters ϑ_k of m_k as shown in Fig.2.

It is worth noting here that since network m_k inherits the parameters of m_{k-1}, the computational effort of loss function $Loss_k$ will be greatly reduced, thus speeding up the training of the network.

2. Multiple rounds of pre-training method. In the first time training PINN, it is not always satisfactory performance of the being trained network due to the randomly parameter initialization. In this case, it is common to conduct training the network in several rounds (pre-training) based on the latest obtained parameters. That is, the obtained parameters after completing the training of an ANN are used as initialization of the network to train again, and so on until the end of the training round. This multi-round pre-training gradually optimizes the parameter initialization, avoids the uncertainty caused by random initialization and guides the improving training quality.

In our PWNN case, the above parameter transfer method (see above the case $k = 1$) is used in the first round of training, and then the multi-round pre-training method is used after the second round. Specifically, each neural network $m_j, j = 2, 3, \cdots, p$ receives the parameters of m_{j-1} as parameter initialization in the first round of training. For subsequent rounds, in training a network, it receives the network parameters obtained from the latest training round of this network as its initialization. This multi-round pre-training will progressively improve the approximation of $\hat{y}(x)$ to the exact solution of IVP (1). To further improve the approximation accuracy of $\hat{y}(x)$ to exact solution $y(x)$, we even apply the two kind techniques interactively in training PWNNs.

4.3 Implementation of PWNN

Let ϑ_k^i denote the training parameter set of network m_k in the ith round and $maxit$ denote the maximum number of iterations. $Loss$ represents the loss value of the neural network, which is determined by Eq.(5), and $\varepsilon > 0$ represents a pre-specified error limitation.

Now, summarizing the above statement of PWNN method, we have the following **Algorithm PWNN** as an implementation of our proposed PWNN method.

Algorithm PWNN

Initialization: Define a generic functional module
Back-propagation=
    {
        While $it \leq maxit \wedge loss \geq \varepsilon$
            Update ϑ_k^i via a optimizer [39]
            Update the loss based on Eq.(5) and back propagation method
    }
Begin:
for $i = 1$ to n **do**
    if $i = 1$ **then**
        for $k = 1$ to p **do**
            if $k = 1$ **then**
                ϑ_k^i is initialized using Xavier method or some others [38]
            else
                initializing $\vartheta_k^i = \vartheta_{k-1}^i$
            end if
                Back-propagation
        end for
    else
        initializing $\vartheta_k^i = \vartheta_k^{i-1}$
        Back-propagation
    end if
end for
End

---

5 Experiment

In this section, we give several numerical experiments using the PWNN algorithm, and compare the results with those of PINN method and Runge-Kutta method to show the validity of our proposed method.

5.1 Example 1

Consider the following IVP of an ODEs

$$\begin{cases} \frac{\mathrm{d}y_1}{\mathrm{d}x} = y_2, \\ \frac{\mathrm{d}y_2}{\mathrm{d}x} = -y_2 - (2 + \sin x)y_1, \\ y_1(0) = 0, y_2(0) = 1, \end{cases} \tag{14}$$

with dependent variable $y(x) = (y_1(x), y_2(x))$ on interval $[0, 10]$. We first use normal PINN to find the ANN solution of the problem. We construct a 3-layer neural network with 1 input neuron, 2 output neurons, and 2 hidden layers of 20 neurons each. We uniformly sampled 1,000 points on the interval $[0, 10]$ as the training data set. The learning rate is set to 0.01, and the number of iterations is 10,000. The training results of the network are shown in Fig. 3.
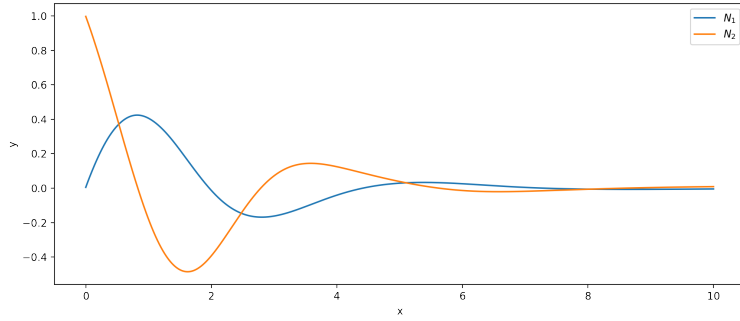
Fig. 3 PINN results for IVP (14) with final loss function value $6.58 * 10^{-5}$.

Then we use PWNN to solve the problem with less training data and the same structure as PINN above on each sub-interval. Interval $[0, 10]$ is divided into five equidistance sub-intervals $\Delta_i = [2(i-1), 2i](i = 1, 2, \cdots, 5)$. The neural network employed in each sub-interval consists of two hidden layers, with each hidden layer containing 20 neurons. In order to compare it with PINN, we uniformly sample 200 data points from each sub-interval as the training data. The learning rate is set to 0.01, and each neural network performed 2000 iterations. The results of training PWNNs $m_i(i = 1, 2, \cdots, 5)$ are shown in Fig. 4. The first five graphs represents the PWNNs on each interval as well as loss function values, and the last graph is their combination $\hat{y}(x) = (\hat{y}_1(x), \hat{y}_2(x))$.

We also solve the problem using the fourth-order Runge-Kutta method to compare the results with those of using above methods.

In Fig. 5, the results obtained by using PINN method, PWNN approach and fourth-order Runge-Kutta method (RK4) are presented. It can be observed that the overall results of the three methods are similar. However, compared to PINN, the results of PWNN and RK4 are closer. It shows that PWNN is more efficiency than PINN in the
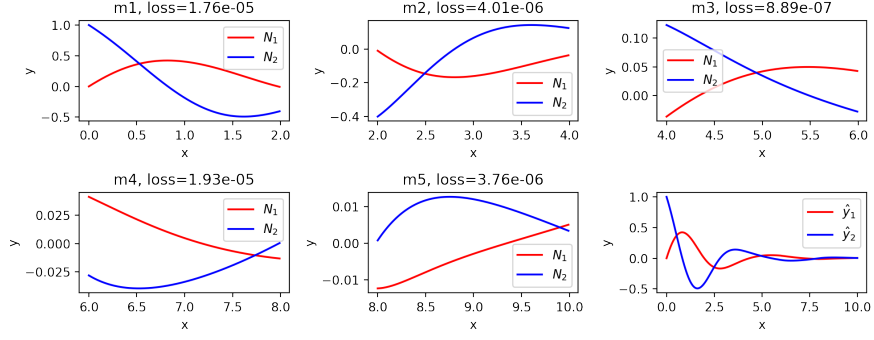
14

Fig. 4 Result figures of PWNNs: the first five is m_k with final loss function value on each sub-interval $\Delta_k(k = 1, 2, \cdots, 5)$. The last one is their combinations $\hat{y}_1(x)$ and $\hat{y}_2(x)$.
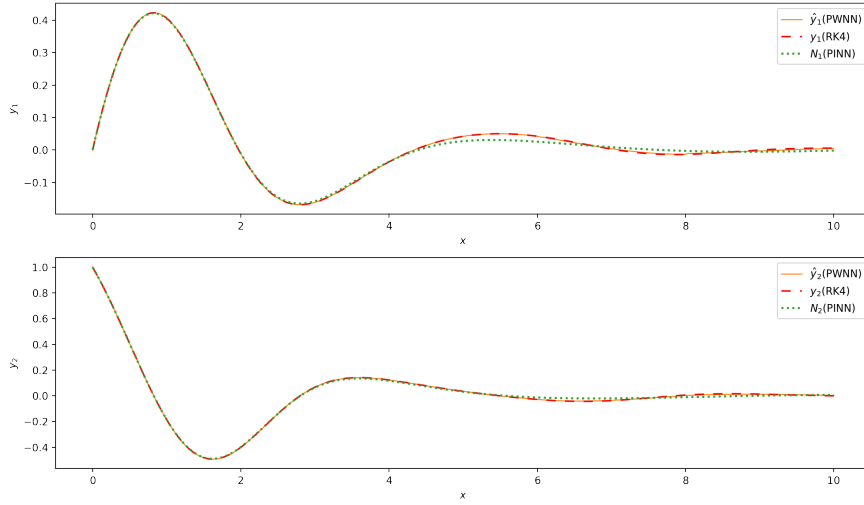

Fig. 5 Comparisons of the results of solving IVP (14) using PINN, PWNN and RK4 methods.

sense of numerical solution.

5.2 Example 2

Consider the following IVP of a SIR model of epidemic dynamics

$$
\begin{cases}
\frac{\mathrm{d}y_1}{\mathrm{d}x} = -0.003 y_1 y_2, \\
\frac{\mathrm{d}y_2}{\mathrm{d}x} = 0.003 y_1 y_2 - 0.1 y_2, \\
\frac{\mathrm{d}y_3}{\mathrm{d}x} = 0.1 y_2, \\
y_1(0) = 98, y_2(0) = 2, y_3(0) = 0.
\end{cases}
\tag{15}
$$

15

First, we use PINN to find its solution over the interval $[0, 50]$. The used PINN consists of 1 input neuron, 3 hidden layers of 20 neurons each, and 3 output neuron. The 200 training points are taken from $[0, 20]$ equidistantly. The network adopts Xavier initialization, employs the Adam optimization algorithm with a learning rate 0.01, and runs for 10000 iterations. Fig. 6 illustrates the solutions given by the PINN for IVP (15) over the interval $[0, 20]$. Observably, the PINN successfully computes solutions merely within the intervals $[0, 14]$ (left graph) instead of large interval $[0, 20]$ (right graph). That is, as the solution interval further extends the learning results of PINN notably deviate from the real case.
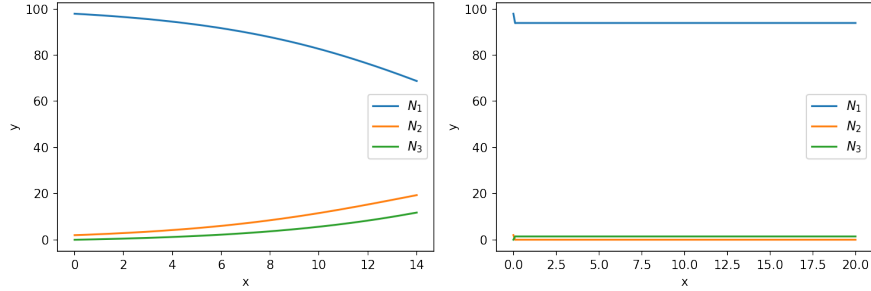
Fig. 6 Results of PINN to solve IVP (15) which failed to get large interval solutions.

We now solve the problem over a large interval $[0, 50]$ using the PWNN method with a smaller size structure than the PINN above. Divide interval $[0, 50]$ into 10 equally sized sub-intervals and Take 100 training points in each sub-interval by uniform distribution. Construct a neural network for each segment, where each network consists of 1 input neuron, 1 hidden layer with 20 neurons, and 3 output neurons. Utilize Xavier initialization and Adam optimization algorithms for each network with learning rate 0.01 and 10000 iterations to train m_1 on interval $[0, 5]$. Fig. 7 illustrates the training results of PWNNs for all sub-intervals. The titles of the figures specify the names of the networks for each interval along with the respective values of the loss function after the completion of network training. Fig. 8 displays the comparisons between the results of both PWNNs and the RK4 method. The solid lines represent the solutions $\hat{y}(x)$ obtained from PWNNs, while the discrete points depict the solutions obtained using RK4 method. Obviously, PWNN gives a large interval solution of problem (15) that is highly consistent with RK4.

Example 2 shows that direct application of conventional PINN can only obtain the inter-cell solution (local solution) of the problem, while PWNN directly gives a large interval solution consistent with RK4, which demonstrates the effectiveness of the given PWNN algorithm.
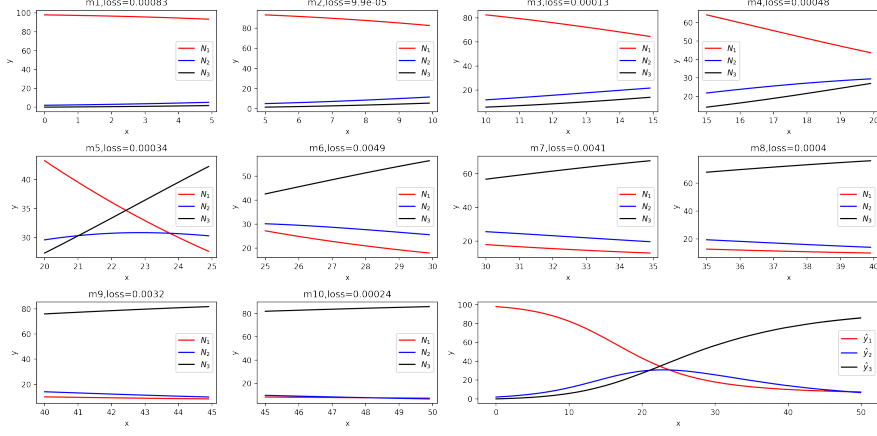
16

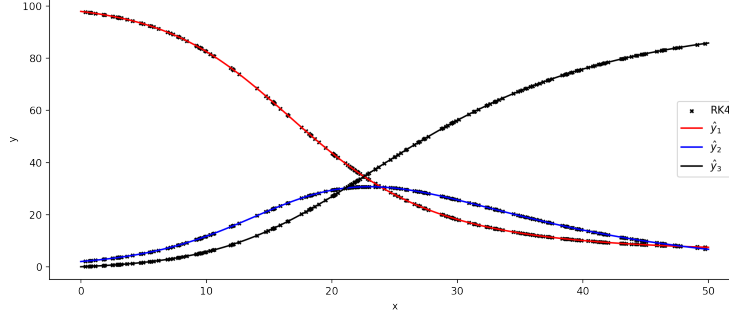**Fig. 7** Results of PWNNs m_i on Δ_i for $(i = 1, 2, \cdots, 10)$ and loss function values for each PWNN.

Fig. 8 Comparison of the results of using PWNN and RK4 methods.

5.3 Example 3

Consider IVP of an ODEs

$$\begin{cases} \frac{\mathrm{d}y_1}{\mathrm{d}x} = \cos x, \\ \frac{\mathrm{d}y_2}{\mathrm{d}x} = -2\sin 2x, \\ y_((0) = 0, \ y_2(0) = 1. \end{cases} \tag{16}$$

It is easy to obtain the analytic solutions of the IVP as $y_1 = \sin x, y_2 = \cos 2x$ for all $x \in R$. Although this example is very simple, it is not easy for PINN to solve this problem over a large interval due to the periodic oscillation of the solutions. We performed four rounds of pre-training for PINN on the problem, with 10000 iterations per round. The training results are shown in Fig.9. It can be observed that despite executing four rounds of training, there is still a considerable discrepancy between the

17

training results and the analytic solutions, especially in the interval $[25, 50]$, where the network did not effectively learn the solutions.

By PWNN method, we divide solution interval $[0, 50]$ into five equal sub-intervals and construct the corresponding PWNNs. These networks were also trained in four rounds of pre-training with same iteration number per sub-interval in each training round. The training results are shown in Fig.10. It can be seen that the PWNNs approximate the analytic solutions quite well after the first training round.
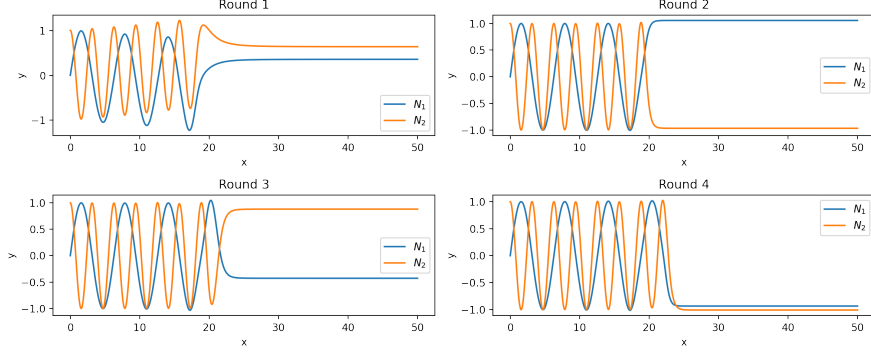

Fig. 9 Results of PINN for IVP (16) which is failure to obtain large interval solutions.
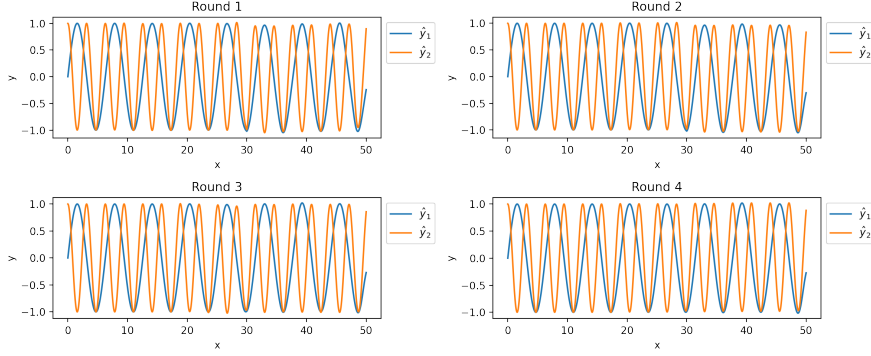

Fig. 10 PWNN solutions to problem (16) over [0, 50]

In the experiments, we use uniformly sampled 100 training points in each interval. Table 1 provides detailed comparisons of convergence trends of PINN and PWNN in terms of decreasing of the loss function values during the four training rounds, under same calculation environment. The loss value for the PWNNs refers to the average of the loss function values in each interval. As can be seen from the table, PWNN has obvious advantages in convergence compared with PINN. In Table 2, we present the convergent trend of PWNN as increasing the number of pre-trining rounds, which

18

shows that the convergence of PWNN is improved by significantly decreasing of loss values.

Table 1 Comparison the convergent trends of PWNN and PINN in terms of loss function values.

Rounds	PINN	PWNN
1	1.59	7.88e-3
2	1.49	5.14e-3
3	1.42	2.93e-3
4	1.34	1.92e-3

Table 2 Convergence of PWNNs in each training round.

loss ＼ PWNN Rounds	m_1	m_2	m_3	m_4	m_5	Mean value of loss
1	1.19e-4	6.32e-4	1.68e-2	3.25e-3	1.86e-2	7.88e-3
2	1.68e-5	3.42e-4	1.57e-2	1.48e-3	8.18e-3	5.14e-3
3	9.85e-6	2.78e-4	5.66e-3	1.08e-3	7.63e-3	2.93e-3
4	6.75e-6	2.53e-4	8.23e-2	9.55e-4	7.65e-3	1.92e-3

5.4 Example 4

Consider the following IVP

$$\begin{cases} \frac{\mathrm{d}y_1}{\mathrm{d}x} = y_2 y_3, \\ \frac{\mathrm{d}y_2}{\mathrm{d}x} = -y_1 y_3, \\ \frac{\mathrm{d}y_3}{\mathrm{d}x} = -0.51 y_1 y_2, \\ y_1(0) = 0, y_2(0) = y_3(0) = 1. \end{cases} \tag{17}$$

As previous examples, we first use PINN to find the numerical solutions of the problem in the interval $[0, 20]$. We perform four rounds of training for PINN, with 10000 iterations per round. The learning rate is set to 0.001 for the first round and 0.0001 for the subsequent rounds. In these experiments, the values of the loss function remains around $1.99 * 10^{-3}$ for different rounds. This indicates that further increasing the number of training rounds and iterations does not significantly reduce the loss function value. The training results are shown in Fig. 11.

In using PWNN, we divide the solution interval $[0, 20]$ into five equal parts and construct corresponding PWNNs. Then, train each PWNN for four rounds, with each round consisting of 10,000 iterations on each sub-interval. The learning rate for the first round of training is set to 0.001, while for the remaining rounds, it is set to 0.0001. The training results of the PWNNs are shown in Fig. 12. The values of the loss function corresponding to each round of PWNN are shown in Table 3. It can be observed that as the number of training rounds increases, the loss function values of
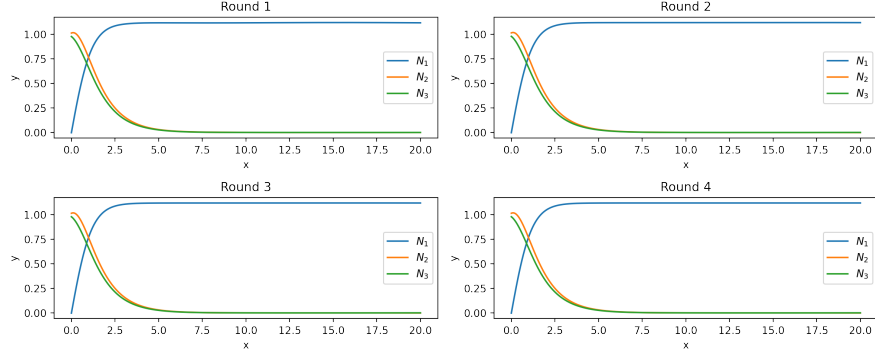
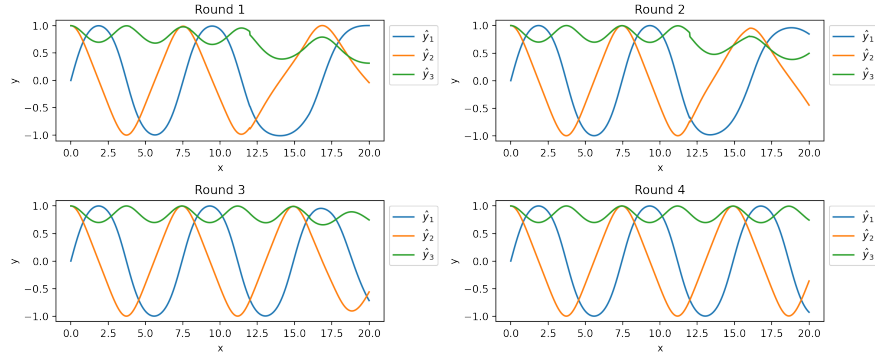**Fig. 11** PINN solutions of problem (17) in different rounds

Fig. 12 PWNN solutions of IVP (17) in different rounds

the PWNNs gradually decreases. In Fig. 13, we show the comparisons of results of the PWNN in the fourth round with the those of RK4. It can be seen that the results of the PWNN method are in good agreement with those of the RK4 method.

Table 3 Comparisons the results obtained by PWNN and RK4 for solving IVP (17).

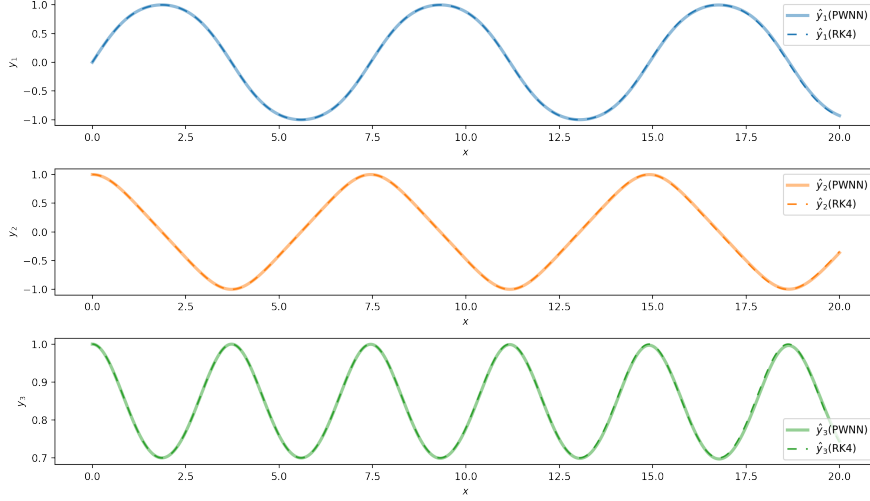| loss ⟍ PWNN Rounds	m_1	m_2	m_3	m_4	m_5
1	1.41e-5	2.17e-4	2.38e-4	3.74e-3	1.30e-4
2	1.53e-6	1.67e-5	1.50e-5	5.28e-3	1.73e-4
3	6.98e-7	4.97e-6	5.68e-6	6.44e-5	1.71e-4
4	4.43e-7	2.20e-6	4.09e-6	1.94e-5	3.61e-5

20

Fig. 13 Comparison of solving results of PWNN and RK4 methods for IVP (17)

6 Discussion and Conclusion

The neural network-based method for solving differential equations provides solutions with excellent generalization properties. Models based on neural networks offer opportunities to theoretically and practically tackle with differential equation problems across various sciences and engineering applications. While most other techniques offer discrete solutions or solutions with limited differentiability, or strongly depend on the discrete scheme of the variable domain. Effectively using the advantages of nonlinear approximation of artificial neural network and overcoming the shortcomings of traditional numerical methods is a hot research direction at present.

In this paper, based on the advantages of PINN local strong convergence, we propose a piecewise neural network method for solving initial value problems of ODEs over large intervals of the independent variable. This method not only provides an approach for coordination of multiple networks to solve an IVP of ODEs but also introduces a parameter transfer and multiple rounds of pre-training technique to effectively enhance the accuracy of network solutions in training multi-correlation ANN models. On one hand, we offer a new method to address the solution extension problem of initial value problems of differential equations. On the other hand, our work aims to contribute to both the theoretical and practical aspects of applying ANN, providing deep insights into the application of modern methods to traditional mathematical problems.

Through comparative experiments, we prove that under almost the same network training environment, the training time of piecewise neural network is shorter, the convergence speed to the solution of the studied problem is faster, and it can also make up for the defects of the basic ANN algorithm. For example, an interesting observation is that when PINN is used to solve the system of equations (16), the network seems to achieve better training performance on the interval $[0, 20]$, while it seems to struggle

21

on the interval $[25, 50]$. This phenomenon reflects PINN's tendency to solve the IVP locally rather than the entire interval. Furthermore, in section 5.2, when PINN is used to solve the equation (15) over a large solution domain, although the final loss value obtained by PINN training is very low, the approximate solution obtained does not match the actual solution. This may reflect two reasons. On the one hand, it shows that the optimization of the residual loss function is only a necessary condition for the artificial neural network output to be an approximate solution of the problem, but not a sufficient condition. On the other hand, this is due to small numerical instabilities during back-propagation because of the complexity of the loss hypersurface where the ANN can settle on a local minimum with a small value for the loss function [2]. While the piecewise neural network method proposed in this paper solves these problems effectively to some extend.

Certainly, there are some limitations to the study. We did not consider applying the proposed method to partial differential equation problems, which would involve partitioning high-dimensional domains of independent variables. The successful application of this method strongly depends on the effective training of sub-neural networks on each corresponding interval. In other words, if an ANN in a certain link cannot be effectively trained in the training of a PWNN, the whole algorithm may not complete the solving task. Furthermore, the continuity and differentiability of network solution $\hat{y}(x)$ over the entire interval have yet to be guaranteed theoretically. These are topics for further study in the future.

References

[1] Braun, M.: Differential Equations and Their Applications. Springer, New York Berlin Heidelbeg Tokyo (1983). https://doi.org/10.1007/978-1-4684-0173-8

[2] Piscopo, M.L., Spannowsky, M., Waite, P.: Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions. Phys. Rev. D **100**, 016002 (2019) https://doi.org/10.1103/PhysRevD.100.016002

[3] Yadav, N., Yadav, A., Kumar, M.: An Introduction to Neural Network Methods for Differential Equations. Springer Briefs in Applied Sciences and Technology. Springer, Springer Dordrecht Heidelberg New York London (2015). https://doi.org/10.1007/978-94-017-9816-7

[4] Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5), 359–366 (1989) https://doi.org/10.1016/0893-6080(89)90020-8

[5] Hornik, K., Stinchcombe, M., White, H.: Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Networks **3**(5), 551–560 (1990) https://doi.org/10.1016/0893-6080(90)90005-6

[6] Li, X.: Simultaneous approximations of multivariate functions and their derivatives by neural networks with one hidden layer. Neurocomputing **12**(4), 327–343

(1996) https://doi.org/10.1016/0925-2312(95)00070-4

[7] Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. **9**(5), 987–1000 (1998) https://doi.org/10.1109/72.712178

[8] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. CoRR **abs/1912.01703** (2019) https://doi.org/10.48550/arXiv.1912.01703 1912.01703

[9] Rall, L.B.: Automatic Differentiation: Techniques and Applications. Springer, New York Berlin Heidelbeg (1981). https://doi.org/10.1007/3-540-10861-0

[10] Verma, A.: An introduction to automatic differentiation. Current Science **78**(7), 804–807 (2000)

[11] Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research **18**(153), 1–43 (2018)

[12] Sirignano, J., Spiliopoulos, K.: Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics **375**, 1339–1364 (2018) https://doi.org/10.1016/j.jcp.2018.08.029

[13] Anitescu, C., Atroshchenko, E., Alajlan, N., Rabczuk, T.: Artificial Neural Network Methods for the Solution of Second Order Boundary Value Problems. Computers, Materials & Continua **59**(1), 345–359 (2019) https://doi.org/10.32604/cmc.2019.06641

[14] 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. Journal of Computational Physics **378**, 686–707 (2019) https://doi.org/10.1016/j.jcp.2018.10.045

[15] Yuan, L., Ni, Y.-Q., Deng, X.-Y., Hao, S.: A-PINN: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. Journal of Computational Physics **462**, 111260 (2022) https://doi.org/10.1016/j.jcp.2022.111260

[16] Chiu, P.-H., Wong, J.C., Ooi, C., Dao, M.H., Ong, Y.-S.: CAN-PINN: A fast physics-informed neural network based on coupled-automatic–numerical differentiation method. Computer Methods in Applied Mechanics and Engineering **395**, 114909 (2022) https://doi.org/10.1016/j.cma.2022.114909

[17] Huang, Y., Hao, W., Lin, G.: HomPINNs: Homotopy physics-informed neural networks for learning multiple solutions of nonlinear elliptic differential equations. Computers & Mathematics with Applications **121**, 62–73 (2022) https://doi.org/10.1016/j.camwa.2022.07.002

[18] Fang, Y., Wu, G.-Z., Wang, Y.-Y., Dai, C.-Q.: Data-driven femtosecond optical soliton excitations and parameters discovery of the high-order NLSE using the PINN. Nonlinear Dynamics **105**(1), 603–616 (2021) https://doi.org/10.1007/s11071-021-06550-9

[19] Bai, Y., Chaolu, T., Bilige, S.: The application of improved physics-informed neural network (IPINN) method in finance. Nonlinear Dynamics **107**(4), 3655–3667 (2022) https://doi.org/10.1007/s11071-021-07146-z

[20] Meng, X., Li, Z., Zhang, D., Karniadakis, G.E.: Ppinn: Parareal physics-informed neural network for time-dependent pdes. Computer Methods in Applied Mechanics and Engineering **370**, 113250 (2020) https://doi.org/10.1016/j.cma.2020.113250

[21] Long, Z., Lu, Y., Dong, B.: Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. Journal of Computational Physics **399**, 108925 (2019) https://doi.org/10.1016/j.jcp.2019.108925

[22] Zha, W., Zhang, W., Li, D., Xing, Y., He, L., Tan, J.: Convolution-based model-solving method for three-dimensional, unsteady, partial differential equations. Neural Computation **34**(2), 518–540 (2022) https://doi.org/10.1162/neco_a_01462

[23] Gao, H., Sun, L., Wang, J.-X.: Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. Journal of Computational Physics **428**, 110079 (2021) https://doi.org/10.1016/j.jcp.2020.110079

[24] Wang, N., Zhang, D., Chang, H., Li, H.: Deep learning of subsurface flow via theory-guided neural network. Journal of Hydrology **584**, 124700 (2020) https://doi.org/10.1016/j.jhydrol.2020.124700

[25] Wang, N., Chang, H., Zhang, D.: Theory-guided auto-encoder for surrogate construction and inverse modeling. Computer Methods in Applied Mechanics and Engineering **385**, 114037 (2021) https://doi.org/10.1016/j.cma.2021.114037

[26] Zhang, R.-F., Bilige, S.: Bilinear neural network method to obtain the exact analytical solutions of nonlinear partial differential equations and its application to p-gBKP equation. Nonlinear Dynamics **95**(4), 3041–3048 (2019) https://doi.org/10.1007/s11071-018-04739-z

[27] Zhang, R., Bilige, S., Chaolu, T.: Fractal Solitons, Arbitrary Function Solutions,

Exact Periodic Wave and Breathers for a Nonlinear Partial Differential Equation by Using Bilinear Neural Network Method. Journal of Systems Science and Complexity **34**(1), 122–139 (2021) https://doi.org/10.1007/s11424-020-9392-5

[28] Chien-Cheng Yu, Yun-Ching Tang, Bin-Da Liu: An adaptive activation function for multilayer feedforward neural networks. In: 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. TENCOM '02. Proceedings., pp. 645–650. IEEE, Beijing, China (2002). https://doi.org/10.1109/TENCON.2002.1181357

[29] Dushkoff, M., Ptucha, R.: Adaptive Activation Functions for Deep Networks. Electronic Imaging **28**(19), 1–5 (2016) https://doi.org/10.2352/ISSN.2470-1173.2016.19.COIMG-149

[30] Qian, S., Liu, H., Liu, C., Wu, S., Wong, H.S.: Adaptive activation functions in convolutional neural networks. Neurocomputing **272**, 204–212 (2018) https://doi.org/10.1016/j.neucom.2017.06.070

[31] Wang, S., Yu, X., Perdikaris, P.: When and why PINNs fail to train: A neural tangent kernel perspective. Journal of Computational Physics **449**, 110768 (2022) https://doi.org/10.1016/j.jcp.2021.110768

[32] Xiang, Z., Peng, W., Liu, X., Yao, W.: Self-adaptive loss balanced Physics-informed neural networks. Neurocomputing **496**, 11–34 (2022) https://doi.org/10.1016/j.neucom.2022.05.015

[33] Jagtap, A.D., Kawaguchi, K., Karniadakis, G.E.: Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. Journal of Computational Physics **404**, 109136 (2020) https://doi.org/10.1016/j.jcp.2019.109136

[34] Wen, Y., Chaolu, T.: Learning the nonlinear solitary wave solution of the korteweg–de vries equation with novel neural network algorithm. Entropy **25**(5) (2023) https://doi.org/10.3390/e25050704

[35] Wen, Y., Chaolu, T.: Study of burgers–huxley equation using neural network method. Axioms **12**(5) (2023) https://doi.org/10.3390/axioms12050429

[36] Wen, Y., Chaolu, T., Wang, X.: Solving the initial value problem of ordinary differential equations by lie group based neural network method. PloS one **17**(4), 0265992 (2022) https://doi.org/10.1371/journal.pone.0265992

[37] Wen, Y., Chaolu, T.: Lie group-based neural networks for nonlinear dynamics. International Journal of Bifurcation and Chaos **33**(14), 2350161–12 (2023) https://doi.org/10.1142/S0218127423501614

[38] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward

neural networks. Journal of Machine Learning Research - Proceedings Track **9**, 249–256 (2010)

[39] Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. arXiv. arXiv:1412.6980 [cs] (2017). https://doi.org/10.48550/arXiv.1412.6980