Indian J Pure Appl Math (2026) 57:628–641
https://doi.org/10.1007/s13226-024-00541-3

ORIGINAL RESEARCH

Approximate solution of KdV-Burgers equation using improved PINNs algorithm

Harender Kumar · Neha Yadav

Received: 27 October 2023 / Accepted: 12 January 2024 / Published online: 31 January 2024

Abstract Finding solutions to partial differential equations (PDEs) has long been a challenging endeavor. Despite various proposed methods, there isn't a universal approach capable of solving all types of PDEs. Recently, deep learning methods have emerged as a powerful tool for the solution of PDEs. Among them, the physics-informed neural networks (PINNs) stand out, integrating fundamental physical laws into neural networks to enforce equation dynamics using space-time data. This paper focuses on utilizing an enhanced version of the PINNs algorithm to approximate solutions for the nonlinear KdV-Burgers equation, a well-known nonlinear PDE with applications ranging from explaining wave propagation in elastic tubes filled with fluid to modeling waves in shallow viscous fluids. Our experiments yield promising results, showcased in the numerical section, where we systematically compare the performance of the PINNs method against our improved variant for each problem instance.

Keywords Physics informed neural network · KdV-Burgers equation · Adaptive activation function · L-BFGS algorithm · Chebfun package

1 Introduction

Over the past few years, the rapid expansion of computational and data resources has facilitated the effective utilization of artificial neural networks (ANNs) across various domains. Specifically, within the realm of machine learning, deep learning has led to groundbreaking advancements in diverse fields, such as speech recognition [1], autonomous driving [2], language translation [3], genomics [4], garbage classification [5], image recognition [6], and pattern recognition [7]. Deep learning approaches have demonstrated exceptional performance in solving highly complex problems, especially when deriving an accurate mathematical description of the problem is challenging [8,9]. However, these approaches heavily rely on a substantial amount of training data for success, which can pose challenges in real-world scenarios where obtaining such data is difficult. Scientific researchers are actively exploring methods to enhance the robustness of training models while operating with limited data, recognizing the critical importance of this endeavor in advancing these technologies.

As per the universal approximation theorem, a neural network equipped with an ample number of neurons in hidden layer holds the capability to approximate any continuous function. This signifies its prowess in addressing nonlinear problems. PDEs serve as fundamental models for numerous nonlinear phenomena across scientific

Communicated by K Sandeep.

N. Yadav (✉)
Department of Mathematics & Scientific Computing, National Institute of Technology Hamirpur, Hamirpur, H.P. 177001, India
E-mail: yadavn@nitj.ac.in

H. Kumar
Department of Mathematics, Dr. B. R. Ambedkar National Institute of Technology, Jalandhar, Punjab 144011, India

and engineering domains. These equations find application in diverse fields including hydrodynamics, chemical kinetics, plasma waves, electromagnetic theory, heat, and mass transfer, among others. Consequently, finding solutions for nonlinear physical phenomena often translates into solving the respective PDEs.

Presently, researchers have devised diverse methodologies aimed at determining both numerical and exact solutions for nonlinear PDEs. For instance, I. Wasim, M. Abbas, and M. Amin [10] proposed a hybrid B-spline collocation method for solving the generalized Burgers-Fisher and Burgers-Huxley equations. Another approach by Ali et al. [11] involves an evolutionary numerical method specifically designed for resolving nonlinear singular periodic boundary value problems. Additionally, Ali and Sadat [12, 13] explored various solutions for PDEs through the application of Lie symmetry [12,13].

Despite the array of approaches available for solving PDEs, numerous concerns persist in this domain. Computational requirements tend to escalate with the complexity of the problem, making it challenging to obtain analytical solutions for PDEs. Additionally, many PDEs lack analytical solutions altogether, compounding the difficulty of resolving them. Consequently, finding solutions for PDEs remains a computationally challenging task. To address these challenges, neural networks serve as valuable nonlinear approximators for solving PDEs. Over recent years, researchers have developed multiple deep learning-based approaches [14–24] tailored to solve PDEs and achieved excellent results.

In 2019 [25], M. Raissi, P. Perdikaris, and G.E. Karniadakis introduced the Physics-Informed Neural Networks (PINNs), a deep learning methodology aimed at solving both forward and inverse problems involving nonlinear PDEs. PINNs directly integrate any physical law defined by nonlinear PDEs into a feed-forward neural network. Key attributes of PINNs include being a mesh-free method, leveraging automatic differentiation, and utilizing neural networks' approximation theory. The mesh-free nature of PINNs allows it to address higher-dimensional nonlinear problems effectively. PINNs were primarily introduced to solve two types of problems: data-driven solutions and data-driven discovery of PDEs. Various studies have showcased the effectiveness of PINNs across diverse domains such as stochastic differential equations, biomedical problems, fractional PDEs, integrable systems, and fluid mechanics [26–29]. While PINNs have demonstrated remarkable success in resolving solutions for PDEs, there remain some complex problems where PINNs haven't delivered satisfactory results. Given the significance of PDEs in disciplines like biology, physics, mathematics, and finance, there's a critical need to enhance the effectiveness of the PINNs methodology.

In 1969, C. H. Su and C. S. Gardner proposed a KDV-Burgers equation for a wide class of nonlinear Galilean-invariant systems under weak nonlinearity and long-wavelength approximations [30]. This equation describes various nonlinear phenomena such as the propagation of waves in elastic tubes filled with a viscous fluid [31], propagation of undular bores in shallow water waves [32], and weakly nonlinear plasma waves with specific dissipative effects [33]. In this paper, we study the KDV-Burgers equation, defined as Eq. (1):

$$\Psi_t + p\Psi\Psi_x - q\Psi_{xx} + r\Psi_{xxx} = 0, \quad x \in [a, b], \ t \geq 0 \tag{1}$$

where p, q and r are positive parameters. It depicts long wavelength approximations in which the effects of the nonlinear advection component $\Psi\Psi_x$ are offset by the dispersion term Ψ_{xxx}. In addition, this equation is used to explain wave propagation through an elastic tube filled with fluid [31] and waves in shallow water in viscous fluids [34].

When the parameter $q = 0$, Eq. (1) transitions into Equation (2), which is commonly referred to as the KDV equation.

$$\Psi_t + p\Psi\Psi_x + r\Psi_{xxx} = 0 \tag{2}$$

In 1895, Kortweg and De-Vries proposed a model which plays an important role in Solitons like waves with slight and limited amplitudes in shallow water. This equation (2) arises in other fields like one-dimensional nonlinear lattice [35], fluid mechanics [36], and others.

At the parameter $r = 0$, Eq. (1) transformed into Eq. (3) which is called the Burger equation:

$$\Psi_t + p\Psi\Psi_x - q\Psi_{xx} = 0 \tag{3}$$

Burgers equation arises in various fields such as mathematical modeling of turbulent fluid, the theory of shock waves, dispersion in porous media, gas dynamics, continuous stochastic processes, wave processes in thermo-elastic medium, and so on. Burgers proposed this equation in 1948 to describe certain characteristics of the turbulent fluid in a channel induced by the interaction of the opposing actions of convection and diffusion [37].

Recently, many researchers have paid a huge amount of time to obtain the solution of the KdV-Burgers equation such as : In [33], the authors investigate the steady-state solution in the case when weak plasma shocks

propagate perpendicular to the magnetic field. In [38], the authors investigate that the steady-state solution of this equation is monotonic whenever diffusion overpowers dispersion, and further shocks become oscillatory when dispersion controls diffusion. Some numerical methods have been presented to solve the KdV-Burgers equation such as modified variational iteration algorithm-II [39], a reliable explicit method [40]. Some numerical methods have been presented to solve the KdV equation such as finite volume scheme [41], decomposition method [42], homotopy analysis method [43], and finite difference scheme [44]. Some numerical and analytical methods have been presented to solve the Burgers problem which are available in the literature [45–47].

Considering the significance of the KdV-Burgers equation and the advantages offered by the PINNs method, this study aims to approximate the solution of the KdV-Burgers equation using a modified version of PINNs, namely MPINNs. In MPINNs, we implement the layer-wise adaptive activation function, enhancing the efficacy of PINNs by incorporating a locally adaptive activation function [48]. This adaptive function demonstrates superior learning capacities compared to the classic activation function, notably improving convergence rates and solution accuracy, particularly in the early stages of training. Within the adaptive activation function, we introduce a scalable hyper-parameter that enables optimization for optimal network performance by dynamically altering the topology of the engaged loss function during the optimization process. An advantageous aspect of this adaptive activation function lies in its ability to tune any number of hidden layers by employing an adaptive hyperparameter alongside a scaling factor.

Additionally, the paper is structured as follows: Section 2 elaborates on the methodology, and Section 3 presents numerical simulation and discussion. Finally, Section 4, provides the conclusion of this study.

2 Methodology

To comprehend the functionality of a neural network, we examine a feed-forward ANN (refer to Fig.1) with a depth of L. This network comprises an input layer, $L-1$ hidden layers, where each hidden layer, denoted as the l-th layer, contains N_l neurons, and concludes with an output layer. Each of the hidden layers of the ANN gets an output $Z^{l-1} \in \mathbb{R}^{N_{l-1}}$ from the previously hidden layer, where the sort of linear transformation given by Eq. (4) :

$$\mathcal{L}_l\left(Z^{l-1}\right) = W^l Z^{l-1} + b^l \tag{4}$$

is carried out. Where, $W^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b^l \in \mathbb{R}^{N_l}$ are the weights and bias of the l-th layer, respectively. Before passing the transformed vector as an input to the next following layer, the nonlinear activation function $\sigma(\cdot)$ is applied to each component of it. Further, at the output layer, an identity activation function is applied. Consequently, the final output expression of ANN is provided by Eq. (5) :

$$O_\theta(Z) = \left(\mathcal{L}_L \circ \sigma \circ \mathcal{L}_{L-1} \circ \ldots \circ \sigma \circ \mathcal{L}_1\right)(Z) \tag{5}$$

where Z represent the input, $\theta = \left\{W^l, b^l\right\}_{l=1}^{L}$ are the trainable parameters of NN, '$\circ$' represent the composition operator, O represent the output of NN.

Now, we discuss the adaptive activation function which was originally proposed by Jagtap et al. [48]. The hyper-parameter α is introduced by the authors of [48] in the activation function, given by Eq. (6) :

$$\sigma\left(\alpha \mathcal{L}_l\left(Z^{l-1}\right)\right) \tag{6}$$

Mathematically, a hyper-parameter α, may vary the slope of the activation function, which is an important feature of neural network training. The hyper-parameter α is optimized along with the weights and biases when finding the minimum of loss function. When looking for global minima, the learning rate has a significant influence. If we use a small learning rate it may slowly approach the global minima and also it may increase the computational cost, while a large learning rate can surpass the global minima. For optimization problems, it is standard practice to utilize a low learning rate, which results in slow variation or, more precisely, slow convergence toward the optimized value. It makes intuitive sense to imagine a scale factor $n \geq 1$ multiplied by α, which accelerates convergence towards global minima. Consequently, the adaptive activation function's final form is provided by Eq. (7) :

$$\sigma\left(n\alpha \mathcal{L}_l\left(Z^{l-1}\right)\right) \tag{7}$$

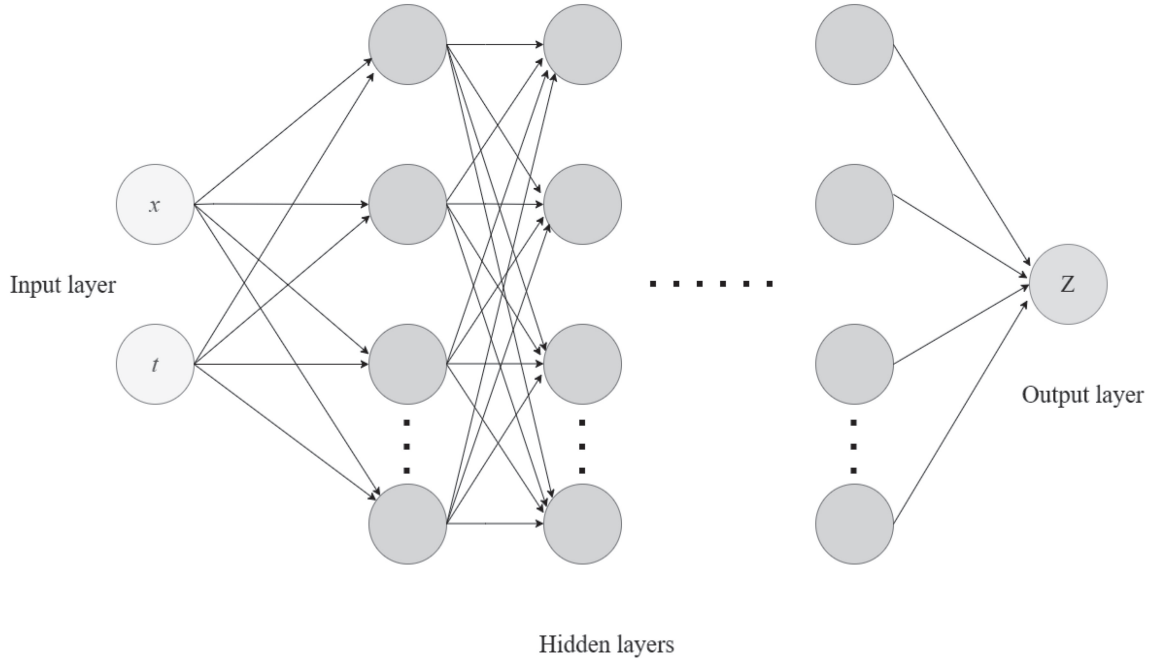

Fig. 1 Feed-forward neural network

We can use this strategy locally i.e. in the adaptive activation function the parameter α could be defined for each hidden layer defined as Eq. (8):

$$\sigma\left(n\alpha^{l}\mathcal{L}_{l}\left(Z^{l-1}\right)\right) \tag{8}$$

it is called the layer-wise adaptive activation function. It is crucial to understand that the previously defined loss function's structure is unaffected by the addition of the scalable hyperparameter. The final representation of the adaptive activation function-based neural network is provided by Eq. (9) :

$$O_{\theta}(Z)=\left(\mathcal{L}_{L}\circ\sigma\circ n\alpha^{L-1}\mathcal{L}_{L-1}\circ\sigma\circ n\alpha^{L-2}\mathcal{L}_{L-2}\circ\ldots\circ\sigma\circ n\alpha^{l}\mathcal{L}_{1}\right)(Z) \tag{9}$$

Now, we briefly introduce PINNs and associated parameters. In PINNs, the physical information described by the PDE is directly imposed on a neural network, while traditional neural network methods are completely dependent on a data-driven methodology that ignores the physical principles that are present in the data. As a result, in order to train neural networks and produce an accurate model, a lot of data is frequently needed. Especially, the PINNs is built to take physical laws into account during training by adding regularization concerning PDEs to the loss function. This strategy reduces the amount of data needed for training and accelerates it.
Recall the KDV-Burger's equation, defined as Eq. (1):

$$\Psi_{t}+p\Psi\Psi_{x}-q\Psi_{xx}+r\Psi_{xxx}=0,\quad x\in[a,b],\ t\geq 0$$

where Ψ is a hidden solution of independent variables (x,t), p,q and r are positive parameters. According to PINNs, we have to construct a physics-informed-neural-network for Eq (1) as follows:

$$f(x,t):=\Psi_{t}+p\Psi\Psi_{x}-q\Psi_{xx}+r\Psi_{xxx} \tag{10}$$

and proceed by approximating $\Psi(x,t)$ by a deep neural network. Now, to optimize the parameters of neural networks $\Psi(x,t)$ and $f(x,t)$, we use mean squared error (MSE) loss defined as follows:

$$C(\theta)=MSE_{\Psi}+MSE_{f} \tag{11}$$

where the MSE is calculated using the following formula:

$$MSE_{\Psi}=\frac{1}{N_{\Psi}}\sum_{i=1}^{N_{\Psi}}\left|\Psi^{i}-\Psi\left(x_{\Psi}^{i},t_{\Psi}^{i}\right)\right|^{2} \tag{12}$$

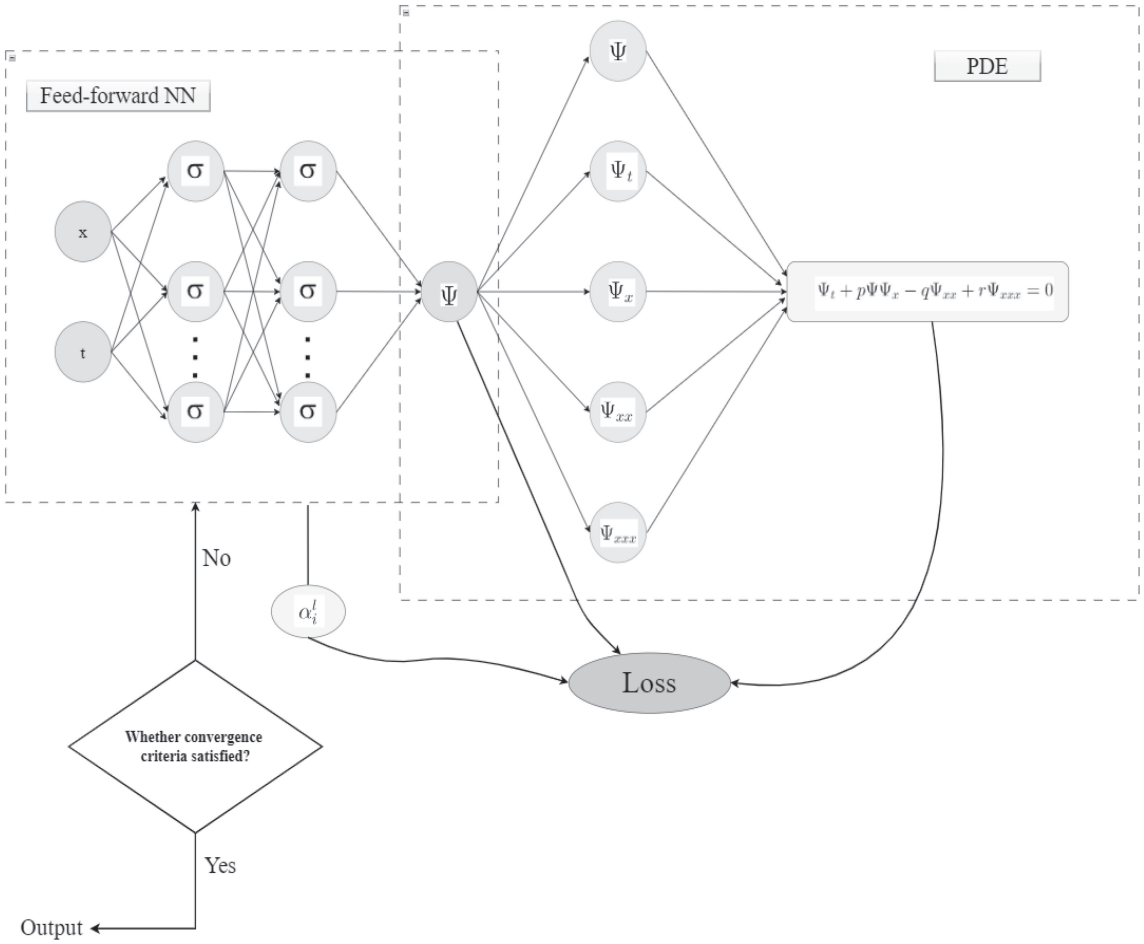

Fig. 2 MPINNs diagram for the KdV-Burgers equation

$$MSE_f = \frac{1}{N_f}\sum_{j=1}^{N_f}\left|f\left(x_f^j, t_f^i\right)\right|^2 \tag{13}$$

Here, $\left(x_\Psi^i, t_\Psi^i, \Psi^i\right)_{i=1}^{N_\Psi}$ represent training data correspond to initial and boundary conditions and $\left(x_f^i, t_f^i\right)_{i=1}^{N_f}$ represent the collocation points for $f(x,t)$. MSE_Ψ represents the loss corresponding to initial and boundary data, and MSE_f represents the loss corresponding to the structure imposed by Eq. (10) at a finite set of collocation points. There are two parts to the training data, in the first part some points N_Ψ are selected by random method from initial and boundary conditions, and in the second part, some points N_f are selected by random method from the time-space region of the domain of the given equation.

In addition, we use PINNs with layer-wise adaptive activation functions and named it MPINNs. Furthermore, in Figure 2 we present the whole process of MPINNs more clearly.

3 Numerical Simulation and discussion

In the subsequent section, we employ the proposed MPINNs method to approximate solutions for three problems associated with the KdV-Burgers equation. We solve all three problems on a larger domain i.e. let $-6 \le x \le 6$ and $0 \le t \le 2$. Specifically, to generate the training data, we divided space $[-6, 6]$ into 256 points and time $[0.0, 2.0]$ into 201 points with the help of Matlab and Chebfun package [49] with a spectral Fourier discretization with 256 modes and a fourth-order explicit Runge time integrator with a time step of 10^{-4}. Thus, for all problems the exact solution $\Psi(x,t)$ is discretized into 51456 points. The neural network model is trained using training

data that consists of the initial conditions, boundary conditions, and some random data in the space-time domain. To demonstrate the advantages of our proposed method, we use the same network depth of neural network as described in the PINNs method and the same training data. There are two parts to the training points, in the first part some points $N_\Psi = 100$ are selected by random method from initial and boundary conditions, and in the second part, some points $N_f = 10,000$ are selected by Latin hypercube sampling method [50] from the time-space region of the domain of given equation. To validate the effectiveness of the MPINNs method, we conduct comparisons between the predicted and exact solutions for each problem. Additionally, we analyze the performance contrast between the proposed MPINNs method and the PINNs method in terms of L_2-error and convergence.

For constructing the neural network, we utilize TensorFlow, a deep learning library equipped with automatic differentiation crucial for information flow within the neural network. Xavier initialization [51] is used to initialize the parameters of the neural network. To generate training points, we adopt the Latin hypercube sampling strategy. The adaptive activation function's scalable parameters are typically initialized as $\alpha_i^l = 0.1$ and $n = 10$. To optimize the loss function, we use a full batch gradient descent optimization method based on the quasi-Newton method called the L-BFGS algorithm [52]. The compilation of code for these problems is executed on an Intel(R) Core(TM) i3-8130U CPU @ 2.20GHz-2.21 GHz machine. We use Anaconda3 which is developed by Peter Wang and Travis Oliphant for the Python 3.6 environment.

Example 1. Consider the Eq. (1) at the values of parameters $p = 1$, $q = 0.1$ and $r = 0.1$, given by Eq. (14):

$$\Psi_t + \Psi\Psi_x - 0.1\,\Psi_{xx} + 0.1\,\Psi_{xxx} = 0 \tag{14}$$

with initial condition given by Eq. (15), boundary conditions are given by Eq. (16) and (17) :

$$\Psi(x,0) = -\frac{6q^2}{25r}\left[1 + \tanh\left(\frac{qx}{10r}\right) + \frac{1}{2}\,\mathrm{sech}^2\left(\frac{qx}{10r}\right)\right] \tag{15}$$

$$\Psi(a,t) = -\frac{6q^2}{25r}\left[1 + \tanh\left(\frac{q}{10r}\left(a + \frac{6q^2}{25r}t\right)\right) + \frac{1}{2}\,\mathrm{sech}^2\left(\frac{q}{10r}\left(a + \frac{6q^2}{25r}t\right)\right)\right] \tag{16}$$

$$\Psi(b,t) = -\frac{6q^2}{25r}\left[1 + \tanh\left(\frac{q}{10r}\left(b + \frac{6q^2}{25r}t\right)\right) + \frac{1}{2}\,\mathrm{sech}^2\left(\frac{q}{10r}\left(b + \frac{6q^2}{25r}t\right)\right)\right] \tag{17}$$

and the exact solution of this equation is given by Eq. (18):

$$\Psi(x,t) = -\frac{6q^2}{25r}\left[1 + \tanh\left(\frac{q}{10r}\left(x + \frac{6q^2}{25r}t\right)\right) + \frac{1}{2}\,\mathrm{sech}^2\left(\frac{q}{10r}\left(x + \frac{6q^2}{25r}t\right)\right)\right] \tag{18}$$

To approximate the solution of Eq. (14), we design a neural network with one input layer, 9 hidden layers each with 20 neurons, and an output layer. In this neural network, we use the Tanh activation function as a layer-wise adaptive activation function. Now, to impose the physical information of the given equation, we construct a physics-informed neural network $f(t,x)$, defined as follows:

$$f(x,t) = \Psi_t + u\Psi_x - 0.1\,\Psi_{xx} + 0.1\,\Psi_{xxx} \tag{19}$$

To obtain the optimal parameters, we continuously optimize the parameters of $\Psi(x,t)$ and $f(x,t)$ by minimizing the mean square error loss given as:

$$C(\theta) = MSE_\Psi + MSE_f$$

MSE_Ψ represents the loss corresponding to initial and boundary data, and MSE_f represents the loss corresponding to the structure imposed by Eq. (14) at a finite set of collocation points.

Following the completion of the neural network model training, we examined the model's effect. The predictions of the neural network model obtained through training are shown in Figure 3, and it can be seen that the predictions obtained are quite refined. To assess the accuracy of this prediction, we pick several moments to compare the prediction with the exact solution. Figure 3 depicts a comparison between the exact solution and the prediction solution at $t = 0.75$, 1.25, and 1.50. Furthermore, the relative L_2 error of this example was determined to be $1.0973E-04$ in 51.23 seconds, confirming the efficiency of the proposed method. While by the PINNs method in 42.364 seconds, the relative L_2 error of this example was determined to be $5.02382E-03$. From Figure 4, we see that PINNs converges in 76 iterations, while MPINNs takes 51 iterations to converge.

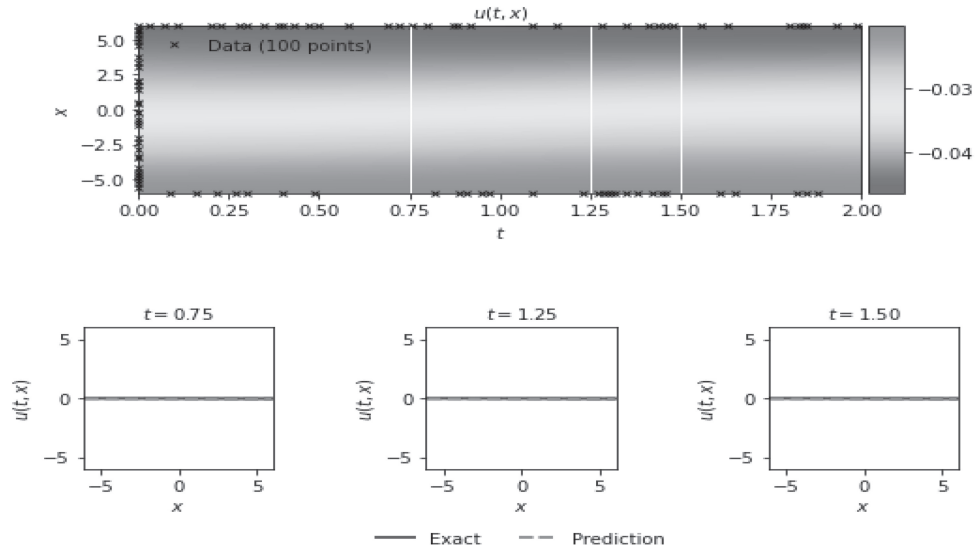

Fig. 3 Comparison between the predicted solution and the exact solution given by MPINNs

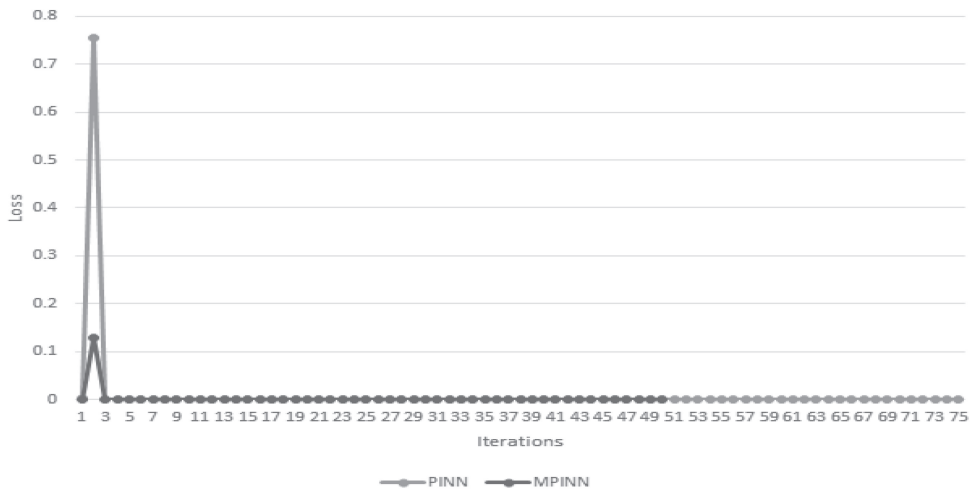

Fig. 4 Comparison of loss functions

Table 1 The relative L_2-norm error between the predicted and exact solution at different combinations of hidden layers and neurons

Neurons	30	40	50
Layers			
3	2.2124E-02	3.2352E-02	3.3256E-02
5	3.4773E-02	3.317E-02	1.4156E-02
7	3.1244E-02	3.4342E-03	4.2563E-03

Table 1 displays an analysis computing the L_2-norm error between the exact and predicted solutions with fixed training points $N_\Psi = 100$ and $N_f = 10,000$ while varying the number of hidden layers and neurons. The findings indicate that as the number of neurons increases with a constant number of hidden layers, the relative error decreases.

In Table 2, an analysis is presented showcasing the L_2-norm error between exact and predicted solutions using a consistent neural network architecture of 7 hidden layers with 20 neurons each, while altering the number of training points (N_Ψ) and collocation points (N_f). The results demonstrate that when the initial boundary data (N_Ψ) is moderate and a substantial number of collocation points (N_f) are employed, the MPINNs approach yields accurate predictive solutions.

Table 2 The relative L2-norm error between the predicted and exact solution with different training points N_Ψ and collocation points N_f

N_f	500	1000	1500	2000
N_Ψ				
25	3.9541E-03	1.3423E-03	3.1476E-03	1.2474E-03
35	2.3652E-03	3.2418E-03	4.2413E-03	4.2446E-03
40	4.4241E-03	4.5414E-03	3.4511E-03	2.5472E-03
50	3.4145E-03	5.4534E-03	3.5121E-03	4.2352E-03

Example 2. Consider the Eq. (1) at the values of parameters $p = 1, q = 0.01$ and $r = 0$, given by Eq. (20):

$$\Psi_t + \Psi\Psi_x - q\Psi_{xx} = 0 \tag{20}$$

with the initial condition given by Eq. (15)

$$\Psi(x, 0) = \frac{\{\epsilon + \mu + (\mu - \epsilon)e^{\nu}\}}{(1 + e^{\nu})} \tag{21}$$

where $\mu = \left(\frac{\epsilon}{q}\right)(x - \eta)$ and ϵ, μ, η, q are the positive parameters. The exact solution of this equation is given by Eq. (22):

$$\Psi(x, t) = \frac{[\epsilon + \mu + (\mu - \epsilon)\exp(\zeta)]}{\{1 + \exp(\zeta)\}} \tag{22}$$

where $\zeta = \left(\frac{\epsilon}{q}\right)(x - \mu t - \eta)$.
To approximate the solution of Eq. (20), we used a neural network with one input layer, 9 hidden layers each with 20 neurons, and an output layer, and Tanh activation function as a layer-wise adaptive activation function. Now, to impose the physical information of the given equation, we construct a physics-informed neural network $f(t, x)$, defined as follows:

$$f(t, x) = \Psi_t + \Psi\Psi_x - 0.01\Psi_{xx} \tag{23}$$

The parameters of $\Psi(t, x)$ and $f(t, x)$ will optimize using mean square error loss defined as:

$$C(\theta) = MSE_\Psi + MSE_f$$

MSE_Ψ represents the loss corresponding to initial and boundary data, and MSE_f represents the loss corresponding to the structure imposed by Eq. (20) at a finite set of collocation points.

Following the completion of the neural network model training, we examined the model's effect. The predictions of the neural network model obtained through training are shown in Figure 5, and it can be seen that the predictions obtained are quite refined. To assess the accuracy of this prediction, we pick several moments to compare the prediction with the exact solution. Figure 5 depicts a comparison between the exact solution and the prediction solution at $t = 0.75$, 1.25, and 1.50. Furthermore, the relative L_2 error of this example was determined to be $3.1457E - 03$ in 37.4061 seconds, confirming the efficiency of the proposed method. While by the PINN method, the relative L_2 error of this example was determined to be $1.184151e - 02$ in 46.2344 seconds. From Figure 6, we see that PINN converges in 214 iterations, while MPINN takes 117 iterations to converge.

In Table 3, we present an analysis to calculate the L_2-norm error value between the exact solution and the predicted solution, when the training points are fixed i.e. $N_\Psi = 100$ and $N_f = 10,000$, and no of hidden layer and no of neurons are different. From Table 3, we can observe that when no of hidden layers are fixed, and no of neurons are increased then relative error is relatively decreased.

In Table 4, we present an analysis to calculate the L_2-norm error value between the exact solution and the predicted solution, when we use a fixed architecture of neural network i.e. 7 hidden layers with 20 neurons in each hidden layer, and no of training points N_Ψ and N_f are different. From Table 4, when the initial boundary data N_Ψ is modest and there are reasonably big collocation points N_f, the MPINN approach can discover accurate prediction solutions.

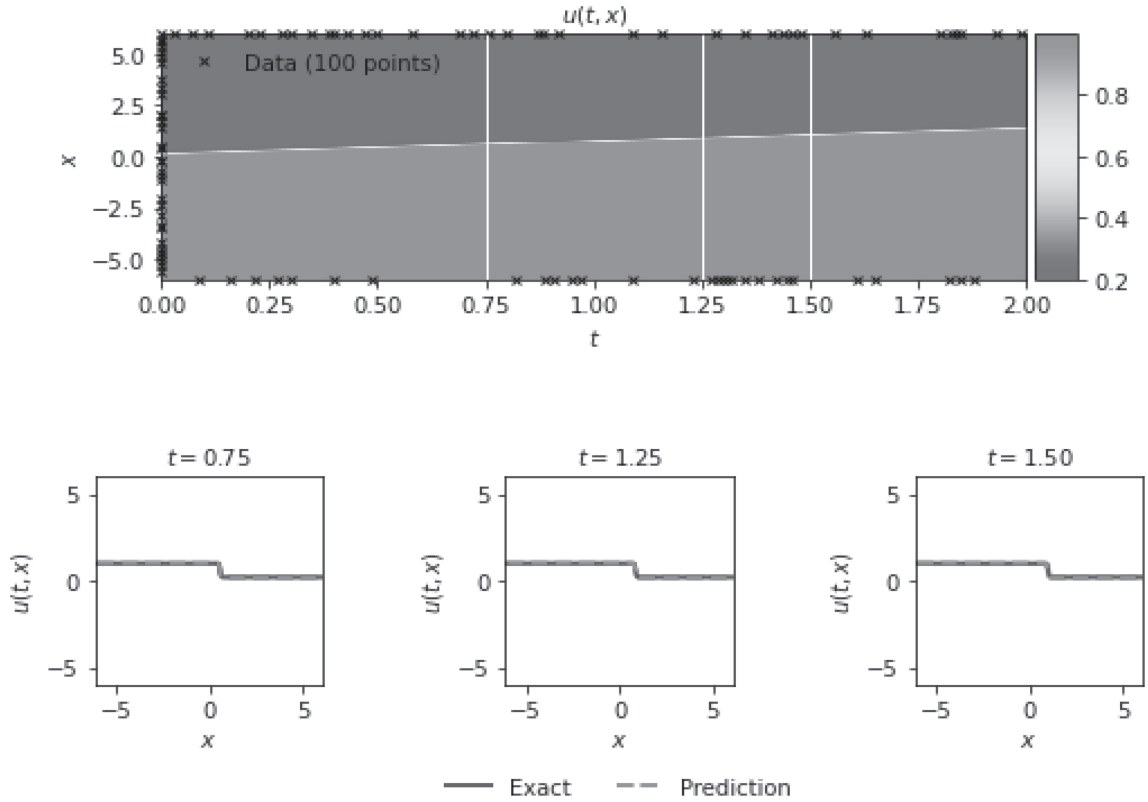

Fig. 5 Comparison between the predicted solution and the exact solution given by MPINN

Example 3 Consider the Eq. (1) at the values of parameters $p = -6$, $q = 0$, and $r = 1$, given by Eq. (24):

$$\Psi_t - 6\Psi\Psi_x + \Psi_{xxx} = 0 \tag{24}$$

with the initial condition given by Eq. (25), boundary conditions are given by Eq. (26) and (27) :

$$\Psi(x, 0) = -2\,\mathrm{sech}^2(x) \tag{25}$$
$$\Psi(a, t) = -2\,\mathrm{sech}^2(a - 4t) \tag{26}$$
$$\Psi(b, t) = -2\,\mathrm{sech}^2(b - 4t) \tag{27}$$

The exact solution of this equation is given by Eq. (28):

$$\Psi(x, t) = -2\,\mathrm{sech}^2(x - 4t) \tag{28}$$

A neural network is used to approximate the solution of Eq. (24), consisting of one input layer, 9 hidden layers each with 20 neurons, and an output layer, as well as Tanh activation function as a layer-wise adaptive activation function. We construct a physics-informed neural network $f(t, x)$ to impose the physical information of the given equation, defined as follows:

$$f(t, x) = \Psi_t - 6\Psi\Psi_x + \Psi_{xxx} \tag{29}$$

To get the optimal parameters of $\Psi(t, x)$ and $f(t, x)$, we continuously optimize the parameters by minimizing the mean square error loss defined as:

$$C(\theta) = MSE_\Psi + MSE_f$$

MSE_Ψ represents the loss corresponding to initial and boundary data, and MSE_f represents the loss corresponding to the structure imposed by Eq. (20) at a finite set of collocation points.

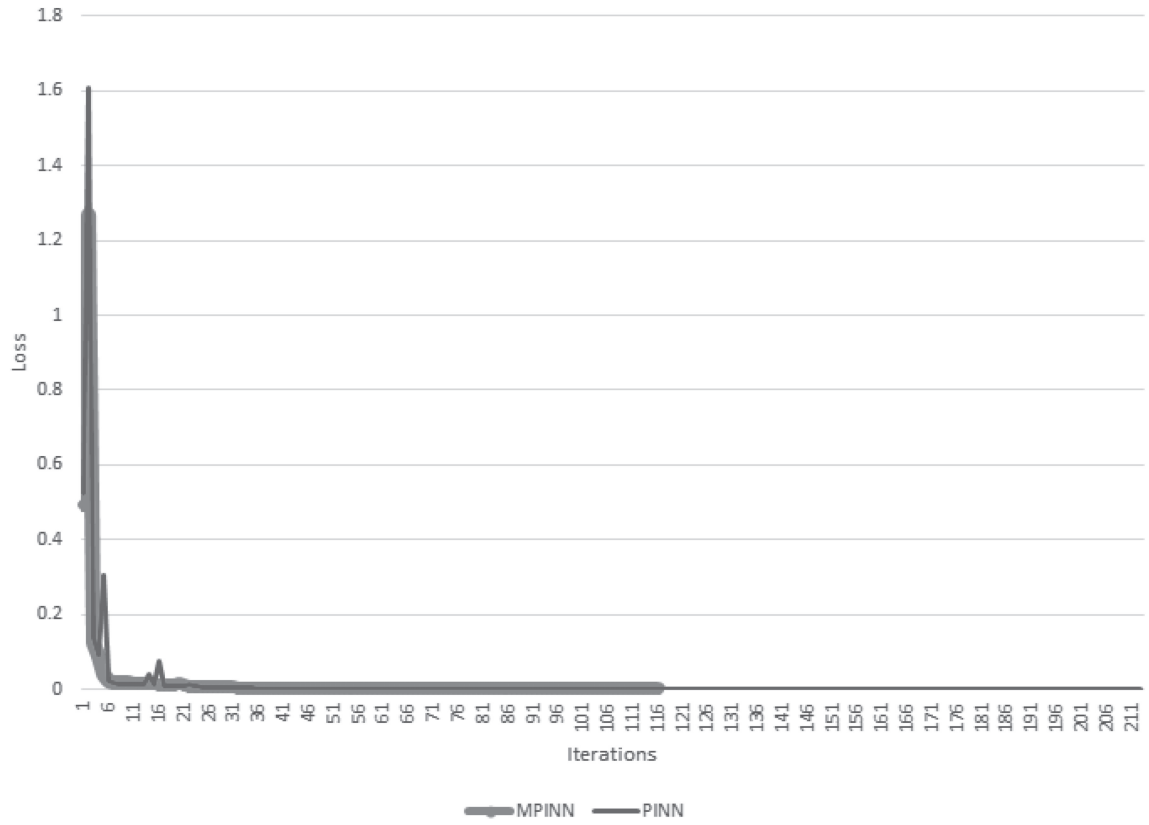

Fig. 6 Comparison of loss functions

Table 3 The relative L_2-norm error between the predicted and exact solution at different combinations of hidden layers and neurons

Neurons	30	40	50
Layers			
3	5.5154E-02	4.4752E-02	7.5456E-02
5	1.4875E-02	1.4575E-02	3.5546E-02
7	7.4644E-02	6.8841E-03	2.4566E-03

Table 4 The relative L2-norm error between the predicted and exact solution with different training points N_Ψ and collocation points N_f

N_f	500	1000	1500	2000
N_Ψ				
25	1.5791E-03	2.4313E-03	5.2706E-03	4.2464E-03
35	4.7054E-03	2.2478E-03	1.4513E-03	3.8446E-03
40	3.5741E-03	2.0804E-03	2.3621E-03	5.4972E-03
50	4.6451E-03	2.7544E-03	4.5371E-03	2.9362E-03

Following the completion of the neural network model training, we examined the model's effect. The predictions of the neural network model obtained through training are shown in Figure 7, and it can be seen that the predictions obtained are quite refined. To assess the accuracy of this prediction, we pick several moments to compare the prediction with the exact solution. Figure 7 depicts a comparison between the exact solution and the prediction solution at $t = 0.75$, 1.25, and 1.50. Furthermore, the relative L_2-error of this example was determined to be $2.11864E-04$ in 313.0023 seconds, confirming the efficiency of the proposed method. While by the PINNs method, the relative L_2-error of this example was determined to be $6.253542E-04$ in 525.24

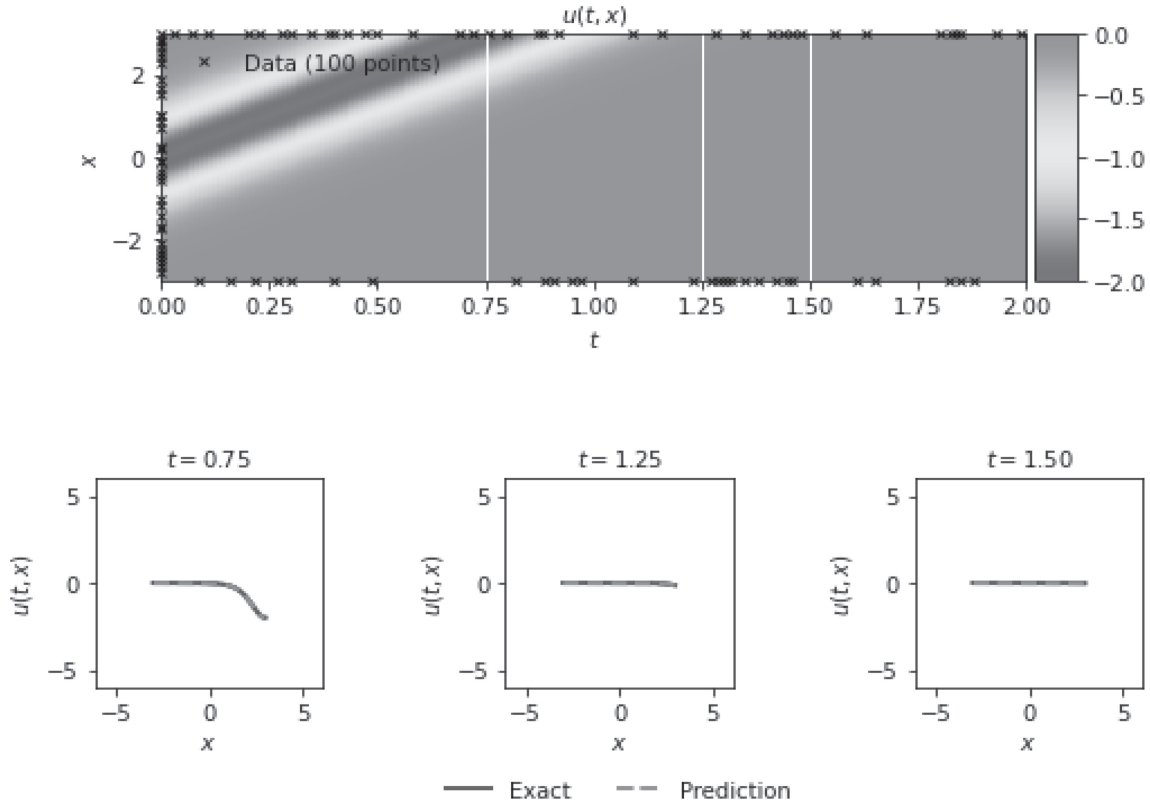

Fig. 7 Comparison between the predicted solution and the exact solution given by MPINN

Table 5 The relative L_2-norm error between the predicted and exact solution at different combinations of hidden layers and neurons

Neurons	30	40	50
Layers			
3	1.1608E-03	2.3113E-03	4.4237E-03
5	2.0223E-03	1.2454E-03	4.5141E-04
7	4.5274E-03	1.0210E-04	2.0259E-04

seconds. From Figure 8, we see that PINN converges in 874 iterations, while MPINN takes 595 iterations to converge.

In Table 5, we present an analysis to calculate the L_2-norm error value between the exact solution and the predicted solution, when the training points are fixed i.e. $N_\Psi = 100$ and $N_f = 10,000$, and no of hidden layer and no of neurons are different. From Table 5, we can observe that when no of hidden layers are fixed, and no of neurons are increased then relative error is relatively decreased.

In Table 6, we present an analysis to calculate the L_2-norm error value between the exact solution and the predicted solution, when we use a fixed architecture of neural network i.e. 7 hidden layers with 20 neurons in each hidden layer, and no of training points N_Ψ and N_f are different. From Table 6, when the initial boundary data N_Ψ is modest and there are reasonably big collocation points N_f, the MPINN approach can discover accurate prediction solutions.

4 Conclusion

In this study, the MPINNs method is proposed to find out the approximate solution of the KdV-Burgers equation. Our experiment shows encouraging results as we compare the results with the PINNs method. In our experiment, the MPINNs method converges fastly in comparison with PINNs, and for each problem, we present an analysis to calculate the L_2-norm error value between the exact solution and the predicted solution, when the training

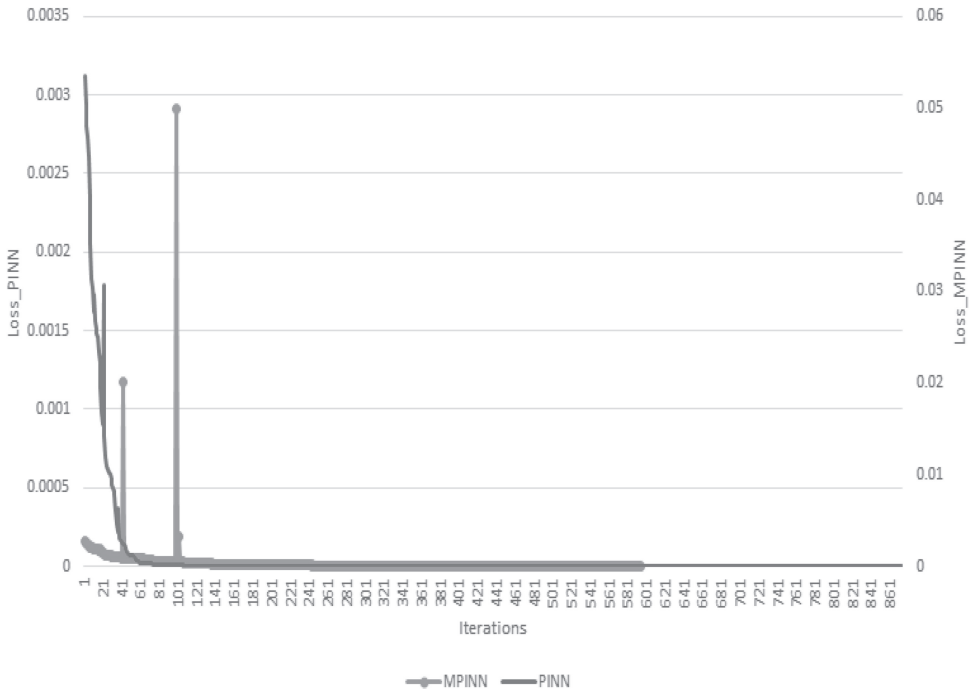

Fig. 8 Comparison of loss functions

Table 6 The relative L_2-norm error between the predicted and exact solution with different training points N_Ψ and collocation points N_f

N_f	500	1000	1500	2000
N_Ψ				
25	1.1768E-03	2.2313e-03	4.4477E-04	1.2471E-04
35	3.04263E-04	4.7184E-04	2.4321E-04	4.2201E-04
40	2.2174E-04	4.4451E-04	2.4192E-04	2.4171E-04
50	2.2654E-04	1.2743E-04	1.2417E-04	1.2416E-04

points are fixed and no of hidden layer and no of neurons are different, we conclude that in our method when no of hidden layers are fixed, and no of neurons are increased then relative error is relatively decreased. Thus, numerical results demonstrate the suggested method's effectiveness, which is dependable and motivating.

Funding The authors declare that they have no funding.

Data Availibility Statement No Data is associated with the manuscript.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Z. Leini, S. Xiaolei, Study on speech recognition method of artificial intelligence deep learning, in: Journal of Physics: Conference Series, Vol. 1754, IOP Publishing, 2021, p. 012183.
2. S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, A survey of deep learning techniques for autonomous driving, Journal of Field Robotics 37 (3) (2020) 362–386.
3. M. N. Y. Ali, M. L. Rahman, J. Chaki, N. Dey, K. Santosh, Machine translation using deep learning for universal networking language based on their structure, International Journal of Machine Learning and Cybernetics 12 (8) (2021) 2365–2376.
4. B. Alipanahi, A. Delong, M. T. Weirauch, B. J. Frey, Predicting the sequence specificities of dna-and rna-binding proteins by deep learning, Nature biotechnology 33 (8) (2015) 831–838.

5. S. Meng, N. Zhang, Y. Ren, X-densenet: deep learning for garbage classification based on visual images, in: Journal of Physics: Conference Series, Vol. 1575, IOP Publishing, 2020, p. 012139.
6. A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012).
7. L. Cai, C. Liu, R. Yuan, H. Ding, Human action recognition using lie group features and convolutional neural networks, Nonlinear Dynamics 99 (2020) 3253–3263.
8. M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear differential equations using gaussian processes, Journal of Computational Physics 348 (2017) 683–693.
9. M. Raissi, P. Perdikaris, G. E. Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, Journal of Computational Physics 335 (2017) 736–746.
10. I. Wasim, M. Abbas, M. Amin, Hybrid b-spline collocation method for solving the generalized burgers-fisher and burgers-huxley equations, Mathematical Problems in Engineering 2018 (2018) 1–18.
11. M. R. Ali, A. R. Hadhoud, W.-X. Ma, Evolutionary numerical approach for solving nonlinear singular periodic boundary value problems, Journal of Intelligent & Fuzzy Systems 39 (5) (2020) 7723–7731.
12. M. R. Ali, R. Sadat, Lie symmetry analysis, new group invariant for the (3+ 1)-dimensional and variable coefficients for liquids with gas bubbles models, Chinese Journal of Physics 71 (2021) 539–547.
13. M. R. Ali, W.-X. Ma, Detection of new multi-wave solutions in an unbounded domain, Modern Physics Letters B 33 (34) (2019) 1950425.
14. R. Guo, Z. Lin, T. Shan, M. Li, F. Yang, S. Xu, A. Abubakar, Solving combined field integral equation with deep neural network for 2-d conducting object, IEEE Antennas and Wireless Propagation Letters 20 (4) (2021) 538–542.
15. J. Jin, L. Zhao, M. Li, F. Yu, Z. Xi, Improved zeroing neural networks for finite time solving nonlinear equations, Neural Computing and Applications 32 (2020) 4151–4160.
16. J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proceedings of the National Academy of Sciences 115 (34) (2018) 8505–8510.
17. Y. Fang, G.-Z. Wu, Y.-Y. Wang, C.-Q. Dai, Data-driven femtosecond optical soliton excitations and parameters discovery of the high-order nlse using the pinn, Nonlinear Dynamics 105 (1) (2021) 603–616.
18. J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial differential equations, Journal of computational physics 375 (2018) 1339–1364.
19. M. A. Nabian, H. Meidani, A deep learning solution approach for high-dimensional random differential equations, Probabilistic Engineering Mechanics 57 (2019) 14–25.
20. W. Cai, Z.-Q. J. Xu, Multi-scale deep neural networks for solving high dimensional pdes, arXiv preprint arXiv:1910.11710 (2019).
21. K. Li, K. Tang, T. Wu, Q. Liao, D3m: A deep domain decomposition method for partial differential equations, IEEE Access 8 (2019) 5283–5294.
22. E. Weinan, B. Yu, The deep ritz method: a deep learning-based numerical algorithm for solving variational problems, Communications in Mathematics and Statistics 6 (1) (2018) 1–12.
23. H. Zhang, Y. Xu, Q. Liu, Y. Li, Deep learning framework for solving fokker–planck equations with low-rank separation representation, Engineering Applications of Artificial Intelligence 121 (2023) 106036.
24. Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, Journal of Computational Physics 411 (2020) 109409.
25. M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational physics 378 (2019) 686–707.
26. Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering 360 (2020) 112789.
27. X. Meng, Z. Li, D. Zhang, G. E. Karniadakis, Ppinn: Parareal physics-informed neural network for time-dependent pdes, Computer Methods in Applied Mechanics and Engineering 370 (2020) 113250.
28. J. Li, Y. Chen, A physics-constrained deep residual network for solving the sine-gordon equation, Communications in Theoretical Physics 73 (1) (2020) 015001.
29. J. Li, Y. Chen, Solving second-order nonlinear evolution partial differential equations using deep learning, Communications in Theoretical Physics 72 (10) (2020) 105005.
30. C. H. Su, C. S. Gardner, Korteweg-de vries equation and generalizations. iii. derivation of the korteweg-de vries equation and burgers equation, Journal of Mathematical Physics 10 (3) (1969) 536–539.
31. R. Johnson, A non-linear equation incorporating damping and dispersion, Journal of Fluid Mechanics 42 (1) (1970) 49–60.
32. R. Johnson, Shallow water waves on a viscous fluid-the undular bore, The Physics of Fluids 15 (10) (1972) 1693–1699.
33. H. Grad, P. N. Hu, Unified shock profile in a plasma, The Physics of Fluids 10 (12) (1967) 2596–2602.
34. A. El-Ajou, O. A. Arqub, S. Momani, Approximate analytical solution of the nonlinear fractional kdv–burgers equation: a new iterative algorithm, Journal of Computational Physics 293 (2015) 81–95.
35. W. Hereman, A. Nuseir, Symbolic methods to construct exact solutions of nonlinear partial differential equations, Mathematics and Computers in Simulation 43 (1) (1997) 13–27.
36. T. Kawahara, Oscillatory solitary waves in dispersive media, Journal of the physical society of Japan 33 (1) (1972) 260–264.
37. J. M. Burgers, A mathematical model illustrating the theory of turbulence, Advances in applied mechanics 1 (1948) 171–199.
38. J. D. Cole, On a quasi-linear parabolic equation occurring in aerodynamics, Quarterly of applied mathematics 9 (3) (1951) 225–236.
39. H. Ahmad, A. R. Seadawy, T. A. Khan, Numerical solution of korteweg–de vries-burgers equation by the modified variational iteration algorithm-ii arising in shallow water waves, Physica Scripta 95 (4) (2020) 045210.
40. S. Ö. Korkut, N. İmamoğlu Karabaş, A reliable explicit method to approximate the general type of the kdv–burgers' equation, Iranian Journal of Science and Technology, Transactions A: Science (2022) 1–11.

41. F. Benkhaldoun, M. Seaid, New finite-volume relaxation methods for the third-order differential equations, Commun. Comput. Phys 4 (820-837) (2008) 3.
42. M. Bektas, M. Inc, Y. Cherruault, Geometrical interpretation and approximate solution of non-linear kdv equation, Kybernetes 34 (7/8) (2005) 941–950.
43. S. Abbasbandy, The application of homotopy analysis method to solve a generalized hirota–satsuma coupled kdv equation, Physics Letters A 361 (6) (2007) 478–483.
44. W. Schiesser, Method of lines solution of the korteweg-de vries equation, Computers & Mathematics with Applications 28 (10-12) (1994) 147–154.
45. J. Biazar, H. Ghazvini, Exact solutions for nonlinear burgers' equation by homotopy perturbation method, Numerical methods for partial differential equations 25 (4) (2009) 833–842.
46. R. Jiwari, R. Mittal, K. K. Sharma, A numerical scheme based on weighted average differential quadrature method for the numerical solution of burgers' equation, Applied Mathematics and Computation 219 (12) (2013) 6680–6691.
47. M. Z. Gorgulu, I. Dag, D. Irk, Wave propagation by way of exponential b-spline galerkin method, in: Journal of Physics: Conference Series, Vol. 766, IOP Publishing, 2016, p. 012031.
48. A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, Journal of Computational Physics 404 (2020) 109136.
49. T. A. Driscoll, N. Hale, L. N. Trefethen, Chebfun guide (2014).
50. M. Stein, Large sample properties of simulations using latin hypercube sampling, Technometrics 29 (2) (1987) 143–151.
51. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
52. D. C. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Mathematical programming 45 (1-3) (1989) 503–528.