Neurocomputing 511 (2022) 131–141

Contents lists available at ScienceDirect

Neurocomputing

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

GCN-FFNN: A two-stream deep model for learning solution to partial differential equations

Onur Bilgin [b], Thomas Vergutz [b], Siamak Mehrkanoon [a,b,*]

[a] Information and Computing Sciences, Utrecht University, Utrecht, Netherlands
[b] Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands

ARTICLE INFO

Article history:
Received 4 May 2022
Revised 7 August 2022
Accepted 4 September 2022
Available online 9 September 2022

Keyword:
Graph convolutional neural network
Partial differential equations
Collocation nodes
Representation learning

ABSTRACT

This paper introduces a novel two-stream deep model based on graph convolutional network (GCN) architecture and feed-forward neural networks (FFNN) for learning the solution of nonlinear partial differential equations (PDEs). The model aims at incorporating both graph and grid input representations using two streams corresponding to GCN and FFNN models, respectively. Each stream layer receives and processes its input representation. As opposed to FFNN which receives a grid-like structure, the GCN stream layer operates on graph input data where the neighborhood information is incorporated through the adjacency matrix of the graph. In this way, the proposed GCN-FFNN model learns from two types of input representations, i.e. grid and graph data, obtained via the discretization of the PDE domain. The GCN-FFNN model is trained in two phases. In the first phase, the model parameters of each stream are trained separately. Both streams employ the same error function to adjust their parameters by enforcing the models to satisfy the given PDE as well as its initial and boundary conditions on grid or graph collocation (training) data. In the second phase, the learned parameters of two-stream layers are frozen and their learned representation solutions are fed to fully connected layers whose parameters are learned using the previously used error function. The learned GCN-FFNN model is tested on test data located both inside and outside the PDE domain. The obtained numerical results demonstrate the applicability and efficiency of the proposed GCN-FFNN model over individual GCN and FFNN models on 1D-Burgers, 1D-Schrödinger, 2D-Burgers, and 2D-Schrödinger equations.

1. Introduction

Partial differential equations (PDEs) are widely used in the mathematical formulation of physical phenomena in a variety of science and engineering applications such as modeling fluid flow, mechanical stress, or material temperature among others. The analytic solutions of PDEs are not often available and therefore several numerical methods such as Finite Difference Methods (FDM) [1], Finite Element Methods [2], splines [3,4], finite volume method [5], Spectral based method [6] have been developed for approximating the solution of the given PDEs.

In particular, in finite difference-based methods, the domain of the PDE is discretized. The solution is only provided for the predefined grid points and additional interpolation is required to obtain the solution for the whole domain. Moreover, the method has a low accuracy in irregular domains, which limits its application in such domains. The finite-element method relies on the discretization of the domain via meshing which can be a challenging and time-consuming process, especially for complex geometries or higher-dimensional PDEs. In addition, similar to finite difference methods, the solution is approximated locally at each mesh point and therefore additional interpolation is required to find the solution at an arbitrary point in the domain [7].

Another class of methods that has been proposed in the literature for the simulation of dynamical systems is based on machine learning approaches and in particular kernel-based models as well as artificial neural networks. The use of neural network-based models for solving ordinary and partial differential equations goes back to the early '90s, see [8–11]. The Hopfield neural networks are used in [9] to solve first-order differential equations. The authors in [12] introduced a feed-forward neural network-based model to solve ordinary and partial differential equations. In their work, the model function is expressed as a sum of two terms where the first term, which contains no adjustable parameters, satisfies

* Corresponding author at: Information and Computing Sciences, Utrecht University, Utrecht, Netherlands.
E-mail addresses: o.bilgin@student.maastrichtuniversity.nl (O. Bilgin), t.vergutz@student.maastrichtuniversity.nl (T. Vergutz), s.mehrkanoon@uu.nl, siamak.mehrkanoon@maastrichtuniversity.nl (S. Mehrkanoon).

https://doi.org/10.1016/j.neucom.2022.09.054
0925-2312/

the initial/boundary conditions and the second term involves a trainable feed-forward neural network. In contrast to mesh-based approaches such as finite difference and finite element methods, neural network models (see [8,13,14]) can generate a closed-form solution and do not require meshing.

Mehrkanoon et al. [15–17,7], for the first time, proposed a systematic machine learning approach based on primal–dual LS-SVM (Least Squares Support Vector Machines) formulation to learn the solution of dynamical systems governed by a range of differential equations including ordinary differential equations (ODEs), partial differential equations (PDEs), differential algebraic equations (DAEs). Unlike the neural network based approaches described in [12,18] that the user had to define a form of a trial solution, which in some cases is not straightforward, in the LS-SVM-based approach the optimal model is derived by incorporating the initial/boundary conditions as constraints of an optimization problem.

In particular, in LS-SVM based model [15,7], the domain of the differential equation is first discretized to generate collocation points that are located inside the domain as well as on its initial and or boundary. Next, one starts with an LS-SVM representation of the solution in the primal and formulates a constrained optimization problem to obtain the optimal values for the model parameters (i.e. weights and biases). More precisely, the initial/boundary conditions of the differential equation are incorporated as constraints of an optimization problem. The formulated constrained optimization problem aims to enforce the LS-SVM representation of the solution to satisfy the given differential equation on the collocation points inside the domain (through the defined objective of the optimization problem) as well as on the initial/boundary of the domain (through the defined hard constraints). One should note that this is not a regression task, as the solution of the differential equation is not provided during the training. In fact, by solving the constrained optimization problem, the optimal representation of the solution is obtained in the dual. The LSSVM code for learning the solution of PDEs (LSSVM-PDE-Solver) is available at [1].

It should also be noted that in the systematic machine learning approach presented in [15,16,7], one can alternatively start with a different representation than the LS-SVM representation; for instance, a neural networks based representation, see [19]. In addition, the hard constraints of the LS-SVM optimization formulation corresponding to the initial/boundary conditions of the PDE can also be relaxed and instead added as an additional term in the objective function of the optimization problem, see [19]. Therefore, motivated by the systematic LS-SVM approach [15,7], the authors in [19] started with a feed-forward neural networks representation and introduced the physics-informed deep learning model and showed its effectiveness in solving differential equations. However, to the best of our knowledge, this existing link between the systematic LS-SVM approach [15,7] for solving differential equations and the physics-informed deep learning model [19] has not been explicitly stated in the literature. Sirignano and Spiliopoulos [20] developed the Deep Galerkin Method (DGM), where the solution of high-dimensional PDE is approximated by a neural network. Zhu et al. [21] developed a dense convolutional encoder-decoder network and E and Yu [22] proposed the Deep Ritz method, based on fully-connected layers and residual connections for solving PDEs.

It is the purpose of this paper to introduce a novel two-stream deep model based on Graph Convolutional Networks (GCNs) and feed-forward neural networks (FFNN) to learn the solution of the given differential equations. GCNs have been successfully applied in many application domains such as natural language processing [23,24], computer vision [25,26] and weather elements forecasting [27,28] tasks. The use of GCNs for learning the solution of PDEs has not been well explored. In the context of PDEs, the grid and graph input data are obtained by discretizing the PDE domain. The proposed model aims to learn from both types of input representations, i.e. grid and graph data. In particular, FFNN processes the grid data, while GCNs operates on graph data and learns the relation between the features by incorporating the neighborhood information through the adjacency matrix of the graph. The highlights of this manuscript can be summarized as follows:

- To introduce a graph convolutional network (GCN) based model to integrate the neighborhood information of the graph data obtained by the discretization of the PDE domain in the learning process.
- To propose a two-stream deep architecture based on the feed forward neural networks (FFNN) and GCN-based model to learn from both grid and graph input representations. This is a step towards the development of multi-view and/or multi-modality learning approaches in learning the solution of dynamical systems.
- To evaluate the proposed model for approximating the solution of 1D-Burgers, 1D-Schrödinger, 2D-Burgers and 2D-Schrödinger equations."

This paper is organized as follows. The proposed model is described in Section 2. The numerical experiments and discussion of the obtained results are given in Section 3. The conclusion is drawn in Section 4.

2. Proposed model

This section first introduces the generation of graph and grid structure data obtained by discretization of the domain of the PDE. Next our two-stream deep neural networks architecture, i.e., GCN-FFNN model, which learns from both grid and graph data as well as the used loss functions are introduced. In Table 1, we list the main symbols and their definitions.

2.1. Graph structure data

The domain of the PDE is first discretized into N nodes. For a 2D-dimensional domain, $N = X \times T$, where X and T are the numbers of elements in space and time dimensions, respectively. We next construct a graph where in particular the neighbors of the (i,j)-th node located inside the domain are $(i-1,j),(i+1,j),(i,j-1)$ and $(i,j+1)$-th nodes. The constructed graphs for equations with two and three-dimensional domains are shown in Fig. 1 (a) and (b), respectively. Here, in the case of two variables each node in the graph has neighbors in two directions x and t, while in the case of three variables nodes have neighbors in three directions x, y, and t. In the constructed graph, let $Z_{\mathscr{B}}$ be the set of all boundary condition nodes, $Z_{\mathscr{I}}$ be the set of all initial condition nodes, and $Z_{\mathscr{B}\mathscr{I}} = Z_{\mathscr{B}} \cup Z_{\mathscr{I}}$. In addition, let Z_D be the set of all nodes except the boundary and initial condition nodes, i.e. all nodes located inside the domain. The cardinality of the above-defined sets $Z_{\mathscr{B}\mathscr{I}}$ and $Z_{\mathscr{D}}$ are denoted by $|Z_{\mathscr{B}\mathscr{I}}|$ and $|Z_{\mathscr{D}}|$.

2.2. Grid structure data

The same discretization steps that were previously used to create the graph data are also used here to generate the grid data points. However, as opposed to previously introduced graph data,

[1] https://github.com/SMehrkanoon/LSSVM-PDE-Solver

Table 1
Summary of the main symbol's definition.

Symbol	Definition
GCN	Graph Convolutional Networks
FFNN	Feed Forward Neural Networks
$Z_{\mathscr{B}}$	Set of all boundary condition nodes
$Z_{\mathscr{I}}$	Set of all initial condition nodes
$Z_{\mathscr{B}\mathscr{I}}$	Union of $Z_{\mathscr{B}}$ and $Z_{\mathscr{I}}$ sets
Z_D	Set of nodes inside the domain
$\widetilde{A}$	Sum of the adjacency and identity matrix
$\widetilde{D}$	Diagonal degree matrix
$H^{(\ell)}$	Matrix of activations in the ℓ-th layer
$W^{(\ell)}$	GCN weights for the ℓ-th layer

the grid data points do not have edge information and only contain the grid data points.

2.3. GCN-FFNN model

Here we propose our two-stream deep neural networks architecture which consists of Graph Convolutional Networks (GCNs) and feed-forward neural networks (FFNN) models in its first and second streams, respectively.

The architecture of the proposed GCN-FFNN model is shown in Fig. 2. The FFNN model shown in the second stream of Fig. 2 was presented in [19,29]. Here, this stream is extended by adding another stream, i.e. the GCN-based model, followed by fully connected layers. The proposed two-stream model incorporates both graph and grid input data in the learning process. In particular, the GCN-based model receives the graph input data while the grid data are fed to the FFNN model. The GCN-based model can integrate the neighborhood information through the adjacency matrix of the graph to improve the prediction accuracy. Therefore, the GCN-FFNN model exploits complementary information of multiple features, i.e., graph and grid data structures, to learn a better representation of the PDE solutions.

The proposed model is trained in two phases. In the first phase, the models in the two streams are trained separately. The same error function is used to adjust the parameters of both streams by enforcing the models to satisfy the given PDE as well as its initial and boundary conditions on grid or graph training data. In the second phase, the learned parameters of two-stream layers are frozen and their learned representation solutions are fed to fully connected layers whose parameters are learned using the previously employed error function. In addition to evaluating the proposed GCN-FFNN model, we have also individually examined each stream, i.e. FFNN as well as GCN-based models. It should be noted that the FFNN-based model has been previously proposed in the literature [19], whereas the GCN-based model and its combination with the FFNN-based model are introduced in this paper. In what follows, the GCN-based model in the GCN-FFNN architecture is explained in more detail.

2.4. GCN-based model

A model based on core Graph Convolutional Network (GCN) [30] is developed and used in the first stream of the proposed GCN-FFNN model to learn the solution of partial differential equations. GCN is an efficient variant of convolutional neural networks which operates directly on graphs. In particular, the following layer-wise propagation rule is utilized in a multi-layer Graph Convolutional Network [30]:

$$H^{(\ell+1)} = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(\ell)}W^{(\ell)}\right). \tag{1}$$

Here, $\widetilde{A}$ is the sum of the adjacency matrix and the identity matrix to include self connections. $\widetilde{D} = \mathrm{diag}(d_1, d_2, \ldots, d_N)$ is the diagonal degree matrix with $d_i = \sum_j \widetilde{A}_{ij}$. In addition, $H^{(\ell)}$ is the matrix of activations in the ℓ-th layer with $H^{(0)}$ equals to the feature representation of the nodes. $W^{(\ell)}$ is the convolution weights for the ℓ-th layer and σ is the activation function, in our case the hyperbolic tangent activation function is used. The model receives the input of the shape $N \times P$, where N is the number of nodes and P denotes the number of node attributes. In our case, for 2-dimensional PDE, $P = 2$ and for 3-dimensional PDE, $P = 3$. Furthermore, the graph structure, i.e. edge information, is also provided to the model through the adjacency matrix. The architecture of our proposed GCN-based model for learning the solutions of PDEs is shown in Fig. 3. The model takes the graph as input. Each node in the graph represents a coordinate point in space and time domain. This graph input is fed to a GCN layer followed by a residual connection and a hyperbolic tangent activation function. The residual part takes also the graph input and has a convolution layer with a filter size of 1×1 to bring the input to the same dimension as the output of the GCN layer. Next, three convolution layers with a filter size of 1×1, each with a hyperbolic tangent activation function are applied. The output of the last convolution layer is followed by a fully-connected layer. We use the systematic machine learning approach presented in [7] to learn the parameters of the model. The proposed GCN-based model receives all the nodes from $Z_{\mathscr{B}\mathscr{I}}$ and $Z_{\mathscr{D}}$ sets as input as well as the partial differential operator $f[\cdot]$

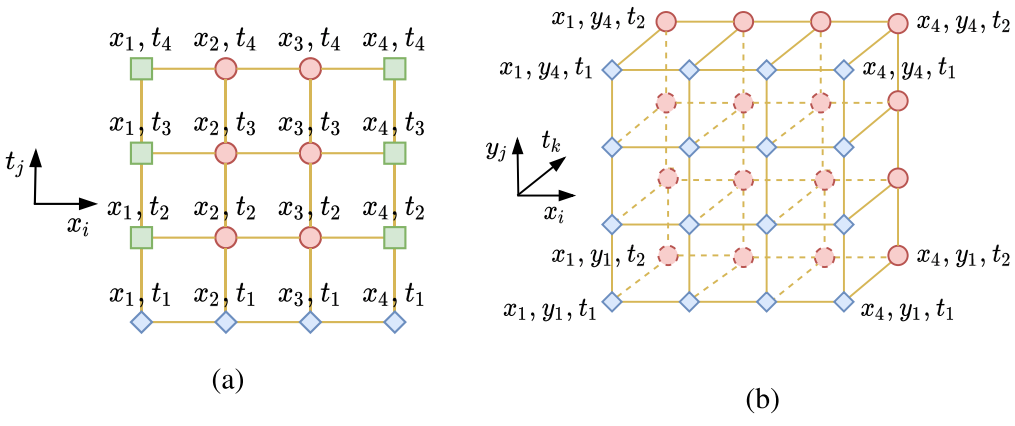

Fig. 1. The used graph structures. (a) nodes with two attributes. (b) nodes with three attributes. Blue nodes are initial conditions and green nodes are boundary conditions. The remaining nodes inside the domain are depicted by red.

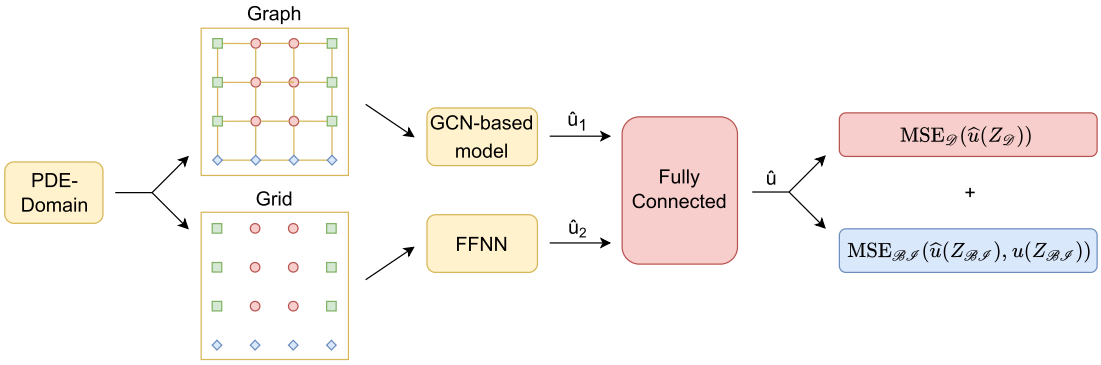

Fig. 2. Architecture of the GCN-FFNN model for learning solutions to PDEs.

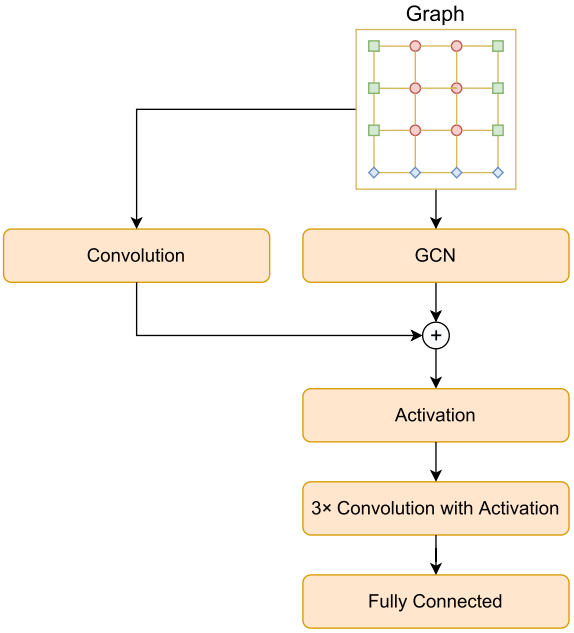

Fig. 3. Architecture of the GCN-based model.

which is obtained by putting all the involved terms of the given PDEs on the left-hand side of the equation so that the right-hand side of the PDE is zero. The model then outputs an approximate solution $\hat{u}$ for the given PDE. The solution $\hat{u}$ is learned by solving an optimization problem whose objective consists of two terms corresponding to the losses defined on inside domain nodes as well as on the initial/boundary condition nodes. The input data to the model consists of the above-mentioned 2- or 3-dimensional nodes and their graph structure (adjacency matrix). It should be noted that here the adjacency matrix is sparse due to the way that the edges of the graph are constructed, see Section 2.1. Here, we use an efficient optimization code available at [31] for dealing with large-scale sparse adjacency matrix in our GCN-layer.

2.5. Loss function

Following the work of Mehrkanoon and Suykens [7], here we use a loss function that enforces the representation of the solution, obtained by our proposed GCN-based model architecture, to satisfy the given differential equations and its initial/boundary conditions. Similar to [7], we aim at minimizing the mean squared loss function to adjust the model parameters. Here, the used loss function is composed of two terms given in Eq. (2). The first term, i.e. $\mathrm{MSE}_{\mathscr{D}}$, enforces the representation of the solution to satisfy the given differential equation inside its domain. The second term, i.e. $\mathrm{MSE}_{\mathscr{B}\mathscr{I}}$, corresponds to making the difference between the true initial/boundary solutions and the model predictions as small as possible. More precisely, given the differential operator $f[\cdot]$ and the collocation nodes from both inside the domain (i.e. $Z^i_{\mathscr{D}}$) as well as on the initial/boundary of the domain (i.e. $Z_{\mathscr{B}\mathscr{I}^i}$), the $\mathrm{MSE}_{\mathscr{D}}$ and $\mathrm{MSE}_{\mathscr{B}\mathscr{I}}$ losses are defined in Eqs. (3) and (4), respectively.

$$\mathrm{MSE}_{\text{total}} = \mathrm{MSE}_{\mathscr{D}} + \mathrm{MSE}_{\mathscr{B}\mathscr{I}}, \tag{2}$$

$$\mathrm{MSE}_{\mathscr{D}} = \frac{\sum_{i=1}^{|Z_{\mathscr{D}}|} f[\hat{u}]\left(Z^i_{\mathscr{D}}\right)}{|Z_{\mathscr{D}}|}. \tag{3}$$

$$\mathrm{MSE}_{\mathscr{B}\mathscr{I}} = \frac{\sum_{i=1}^{|Z_{\mathscr{B}\mathscr{I}}|} \left(u(Z_{\mathscr{B}\mathscr{I}^i}) - \hat{u}(Z_{\mathscr{B}\mathscr{I}^i})\right)^2}{|Z_{\mathscr{B}\mathscr{I}}|}. \tag{4}$$

3. Numerical results

In this section, four experiments are performed to demonstrate the efficiency of the proposed GCN-FFNN model for learning the solution of 1D- and 2D-Burgers equations as well as 1D- and 2D-Schrödinger equations. The model parameters are learned in a transductive fashion. The input dataset is divided into train and test sets. The training set contains nodes from both inside the domain as well as its initial and boundary. Two scenarios are examined for test nodes, i.e. test nodes inside and outside the domain of PDE, see Fig. 4. In the first case, see Fig. 4 (a), some grid points along the x-dimension are first randomly selected and then all the (x,t)-nodes with those selected x-coordinate positions form the inside domain test nodes. It should be noted that for PDEs with 3-dimensional domains, the random grid points are selected along the x- and y-dimensions, and subsequently all the nodes (x,y,t)-nodes with those selected (x,y)-coordinate positions form the inside domain test nodes. In the second case, see Fig. 4 (b), the test nodes are from outside the domain. In both test cases, the test set is 10% of the whole dataset.

The accuracy of the obtained approximate solution is measured by means of mean squared as well as infinite error norms defined as follows:

$$\mathrm{MSE}_{\text{test}} = \frac{\sum_{i=1}^{N_{\text{test}}} e_i^2}{N_{\text{test}}}, \quad \mathrm{L}_\infty = ||e||_\infty. \tag{5}$$

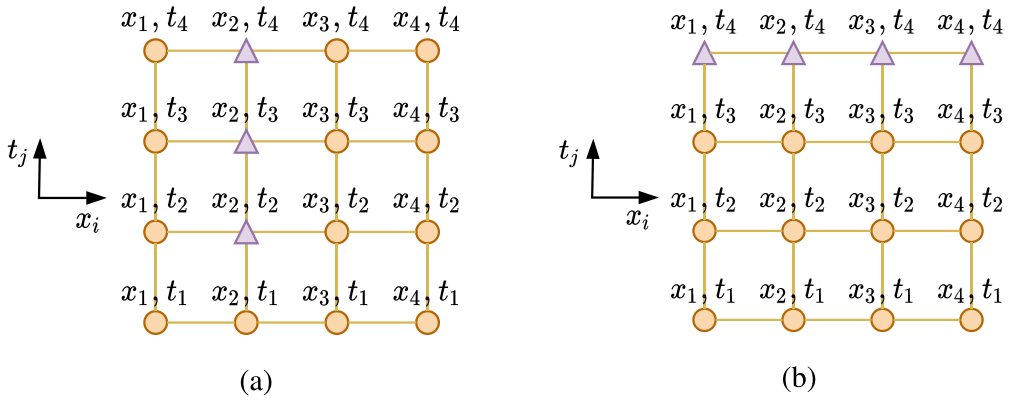

Fig. 4. Two scenarios for selecting the test nodes. Purple triangle nodes are test samples and yellow circle nodes are training samples. (a): The test nodes are from inside the domain. (b): The test nodes are from outside the domain.

Here, $e_i = u\left(Z^i_{\text{test}}\right) - \hat{u}\left(Z^i_{\text{test}}\right)$ where Z^i_{test} is the i-th test node and N_{test} is the number of test nodes. The obtained results of three models, i.e. FFNN, GCN, and GCN-FFNN, on the above-mentioned four equations are compared. All the models are trained using L-BFGS with a learning rate of 1.0 for a maximum of 50000 epochs [32]. To make a fair comparison, all the models receive the same training and test sets. The number of layers, hidden units, and trainable parameters of each of the used modules of GCN-FFNN architecture for each equation are empirically found and tabulated in Table 2.

3.1. 1D-burgers equation

The 1D-Burgers equation with boundary and initial conditions is given in Eq. (6) [19]:

$$\begin{cases} u_t + uu_x - (0.01/\pi)u_{xx} = 0,\ x \in [-1,1],\ t \in [0, 0.99], \\ u(x,0) = -sin(\pi x), \\ u(-1,t) = u(1,t) = 0. \end{cases} \tag{6}$$

Here, the differential operator is defined as $f[u] := u_t + uu_x - (0.01/\pi)u_{xx}$. The solution we seek to approximate with our proposed model is $u(x,t)$ (shown as u in the equation). The variables u_x and u_t are the partial derivatives of u with respect to x and t, respectively. u_{xx} is the second partial derivative of u with respect to x.

In the first scenario, the two dimensional domain of the 1D-Burgers equation, i.e. $(x,t) \in [-1,1] \times [0, 0.99]$, is divided into $N = 256 \times 100$ nodes, which are evenly spaced in each dimension. We have randomly selected 10% of N nodes to form the test nodes inside the domain. In the second scenario, the nodes in the ranges $(x,t) \in [-1,1] \times [0, 0.89]$ are used for training and the test nodes outside the domain are selected from $0.89 < t \leqslant 0.99$. The obtained MSEs and infinity norm of each model are tabulated in Table 3. Both used metrics show that the proposed GCN-FFNN model achieved the best results for both inside and outside test nodes.

Fig. 5 corresponds to the first scenario where the test nodes are from inside the domain. In particular, Fig. 5 (a), (b) and (c) show the true and approximate solution obtained by GCN-FFNN model for the 1D-Burgers equation at $x = -0.15, x = 0.15$ and $x = 0.94$. Here, the prediction at $x = -0.15$ is from training set, whereas the predictions at $x = 0.15$ and $x = 0.94$ are from test set. The obtained residuals are shown in Fig. 5 (d), (e) and (f). Fig. 6 corresponds to the second scenario where the test nodes are from outside the domain. Fig. 6 (a), (b) and (c) show the true and approximate solution obtained by GCN-FFNN model. Here, the predictions at $t = 0.50$ and $t = 0.75$ are from the train set, while the predictions at $t = 0.99$ are from the test set. Fig. 6 (d), (e) and (f) are the residuals at $t = 0.50, t = 0.75$ and $t = 0.99$.

3.2. 1D-Schrödinger equation

The 1D-Schrödinger equation with boundary and initial conditions is described in Eq. (7) [29].

$$\begin{cases} i\psi_t + 0.5\psi_{xx} + |\psi|^2\psi = 0,\ x \in [-5,5],\ t \in [0, \frac{\pi}{2}], \\ \psi(x,0) = 2sech(x) \\ \psi(-5,t) = \psi(5,t) \\ \psi_x(-5,t) = \psi_x(5,t) \end{cases} \tag{7}$$

The differential operator is defined as $f[u] := f = i\psi_t + 0.5\psi_{xx} + |\psi|^2\psi$. Let u and v denote the real and imaginary components of ψ, respectively. Then the 1D-Schrödinger equation can be rewritten as follows [29]:

$$\begin{cases} u_t = -0.5v_{xx} - (u^2 + v^2)v, \\ v_t = 0.5u_{xx} + (u^2 + v^2)u. \end{cases} \tag{8}$$

In the first scenario, the domain of the 1D-Schrödinger equation $(x,t) \in [-5,5] \times [0, \frac{\pi}{2}]$ is divided into $N = 257 \times 201$ nodes, which are evenly spaced for each dimension. Following the previously mentioned approaches for creating the test nodes, 10% of N nodes are randomly selected to form the inside domain test nodes. In the second scenario, the nodes in the ranges $(x,t) \in [-5,5] \times [0, 0.9\frac{\pi}{2}]$ are used for training and the outside domain test nodes are selected from $0.9\frac{\pi}{2} < t \leqslant \frac{\pi}{2}$. The obtained results for the 1D-Schrödinger equation on both inside and outside domain test nodes are shown in Table 3. FFNN outperforms the other models for inside domain test nodes. For the outside domain test nodes, the GCN-FFNN model shows the least MSE error compared to the other models, while its infinity norm error remains higher compared to FFNN.

Fig. 7 corresponds to the first scenario where the test nodes are from inside the domain. In Fig. 7 (a), (b) and (c) the true solution and the obtained results of GCN-FFNN model at $x = -1.17, x = 0$ and $x = 1.56$ are shown, respectively. Here, the prediction at $x = -1.17$ is from training set, whereas the predictions at $x = 0$ and $x = 1.56$ are from test set. The obtained residuals are shown in Fig. 7 (d), (e) and (f). Fig. 8 corresponds to the second scenario where the test nodes are from outside the domain. The true and approximate solutions at $t = 0.11$ $t = 0.80$ and $t = 1.57$ are shown Fig. 8 (a), (b) and (c), respectively. Here, the predictions at $t = 0.11$ and $t = 0.80$ are from the training set, while the predictions at

Table 2
The empirically found hyper-parameters of each module of the GCN-FFNN model for each equations. In the GCN-based model, the GCN layer, convolution layers, and the fully connected layer have the same hidden unit number.

PDE	Modul	# Layers	# Hidden Units	# Trainable Parameters
1D-Burgers	FFNN-based model [19]	8	20	2601
	GCN-based model	1	12	553
	Fully-Connected	2	48	144
1D-Schrödinger	FFNN-based model [29]	6	100	40902
	GCN-based model	1	256	199426
	Fully-Connected	1	1	2
2D-Burgers	FFNN-based model [29]	8	20	2621
	GCN-based model	1	12	577
	Fully-Connected	2	16	48
2D-Schrödinger	FFNN-based model [29]	5	50	7952
	GCN-based model	1	18	1208
	Fully-Connected	2	16	48

Table 3
The obtained MSEs and infinity norm errors for inside and outside test sets.

PDE	Model	Test nodes			
		Inside the domain		Outside the domain	
		MSE_{test}	L_∞	MSE_{test}	L_∞
1D-Burgers	FFNN	$5.10 \cdot 10^{-6}$	0.025	$6.04 \cdot 10^{-6}$	0.029
	GCN	$6.44 \cdot 10^{-4}$	0.139	$8.81 \cdot 10^{-4}$	0.383
	GCN-FFNN	$\mathbf{3.87 \cdot 10^{-6}}$	**0.022**	$\mathbf{1.50 \cdot 10^{-6}}$	**0.019**
1D-Schrödinger	FFNN	$\mathbf{9.00 \cdot 10^{-6}}$	**0.008**	$5.42 \cdot 10^{-5}$	**0.017**
	GCN	$1.30 \cdot 10^{-4}$	0.023	$9.48 \cdot 10^{-4}$	0.030
	GCN-FFNN	$8.75 \cdot 10^{-5}$	0.011	$\mathbf{3.39 \cdot 10^{-5}}$	0.027
2D-Burgers	FFNN	$1.68 \cdot 10^{-3}$	0.085	$4.52 \cdot 10^{-3}$	0.086
	GCN	$2.27 \cdot 10^{-3}$	0.094	$\mathbf{5.92 \cdot 10^{-4}}$	0.030
	GCN-FFNN	$\mathbf{1.49 \cdot 10^{-3}}$	**0.077**	$5.99 \cdot 10^{-4}$	**0.027**
2D-Schrödinger	FFNN	$\mathbf{1.47 \cdot 10^{-7}}$	**0.002**	$3.02 \cdot 10^{-7}$	**0.002**
	GCN	$1.19 \cdot 10^{-6}$	0.003	$1.51 \cdot 10^{-6}$	0.003
	GCN-FFNN	$\mathbf{1.47 \cdot 10^{-7}}$	**0.002**	$\mathbf{2.58 \cdot 10^{-7}}$	**0.002**

$t = 1.57$ are from the outside domain test nodes. The obtained residuals are shown in Fig. 8 (d), (e) and (f).

3.3. 2D-burgers equation

The 2D-Burgers equation with boundary and initial conditions is given in Eq. (9) [29].

$$\begin{cases} u_t + u(u_x + u_y) - 0.1(u_{xx} + u_{yy}) = 0, \ (x,y) \in [0,1], \ t \in [0,3], \\ u(x,y,0) = 1/(1 + \exp[(x + y - t)/0.2]), \end{cases} \tag{9}$$

Here, the differential operator is defined as $f[u] := f = u_t + u(u_x + u_y) - 0.1(u_{xx} + u_{yy})$.

The domain of the 2D-Burgers equation, i.e. $(x,y,t) \in [0,1] \times [0,1] \times [0,3]$, is divided into $N = 26 \times 26 \times 31$ nodes. In the first scenario, 10% of N nodes are randomly selected to form the inside domain test nodes and the remaining nodes are used for training the models. In the second scenario, the nodes in the ranges $(x,y,t) \in [0,1] \times [0,1] \times [0,2.7]$ are used for training and the outside domain test nodes are selected from $2.7 < t \leqslant 3$. From Table 3, one can observe that for the 2D-Burgers equation, the GCN-FFNN model outperforms the other models on inside domain test nodes using both MSE and infinity norms. For outside domain test nodes, GCN outperforms the other models in terms of MSE metric, while the GCN-FFNN model achieved the least infinity norm compared to the other models.

Fig. 9 (a) and (b) show the true solution at $t = 1$ and $t = 3$ from training and test set, respectively. The obtained residuals are shown in Fig. 9 (c) and (d).

3.4. 2D-Schrödinger equation

The 2D-Schrödinger equation with boundary and initial conditions are depicted in Eq. (10) [29]

$$\begin{cases} i\psi_t + \psi_{xx} + \psi_{yy} + w(x,y)\psi = 0, \ (x,y) \in [-5,5], \ t \in [0,1], \\ \psi(x,y,t) = ie^{it}/(cosh(x) + cosh(y)) \end{cases} \tag{10}$$

with

$$w(x,y) = 3 - 2tanh^2(x) - 2tanh^2(y), \tag{11}$$

Here, the differential operator is defined as $f[u] := f = i\psi_t + \psi_{xx} + \psi_{yy} + w(x,y)\psi$. Let u and v denote the real and imaginary components of ψ, then the 2D-Schrödinger equation can be rewritten as follows [29]:

$$\begin{cases} u_t = -v_{xx} - v_{yy} - wv, \\ v_t = u_{xx} + u_{yy} - wu. \end{cases} \tag{12}$$

The domain of the 2D-Schrödinger equation, i.e. $(x,y,t) \in [-5,5] \times [-5,5] \times [0,1]$, is divided into $N = 26 \times 26 \times 11$ nodes. In the first scenario, 10% of N nodes are randomly selected to form the inside domain test nodes and the remaining nodes are used for training the models. In the second scenario, the nodes in the ranges $(x,y,t) \in [-5,5] \times [-5,5] \times [0,0.9]$ are used for training

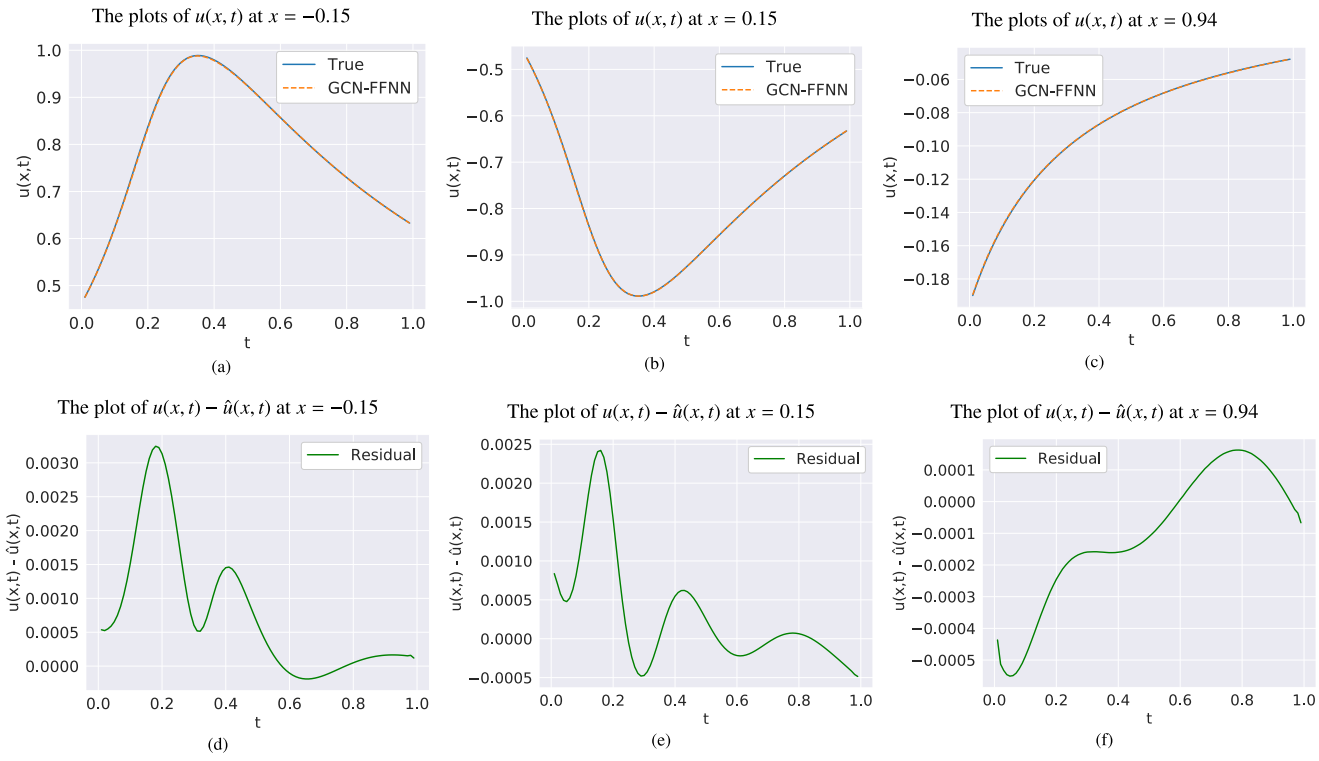

Fig. 5. First scenario: The plots for the 1D-Burgers equation obtained by GCN-FFNN model corresponding to the first scenario where test nodes are from inside the domain. (a), (b) and (c): The true and approximate solution obtained by GCN-FFNN model. (d), (e) and (f): The obtained residuals $u(x,t) - \hat{u}(x,t)$. The data at $x = -0.15$ belongs to the training set, while the data at $x = 0.15$ and $x = 0.94$ are from test set.

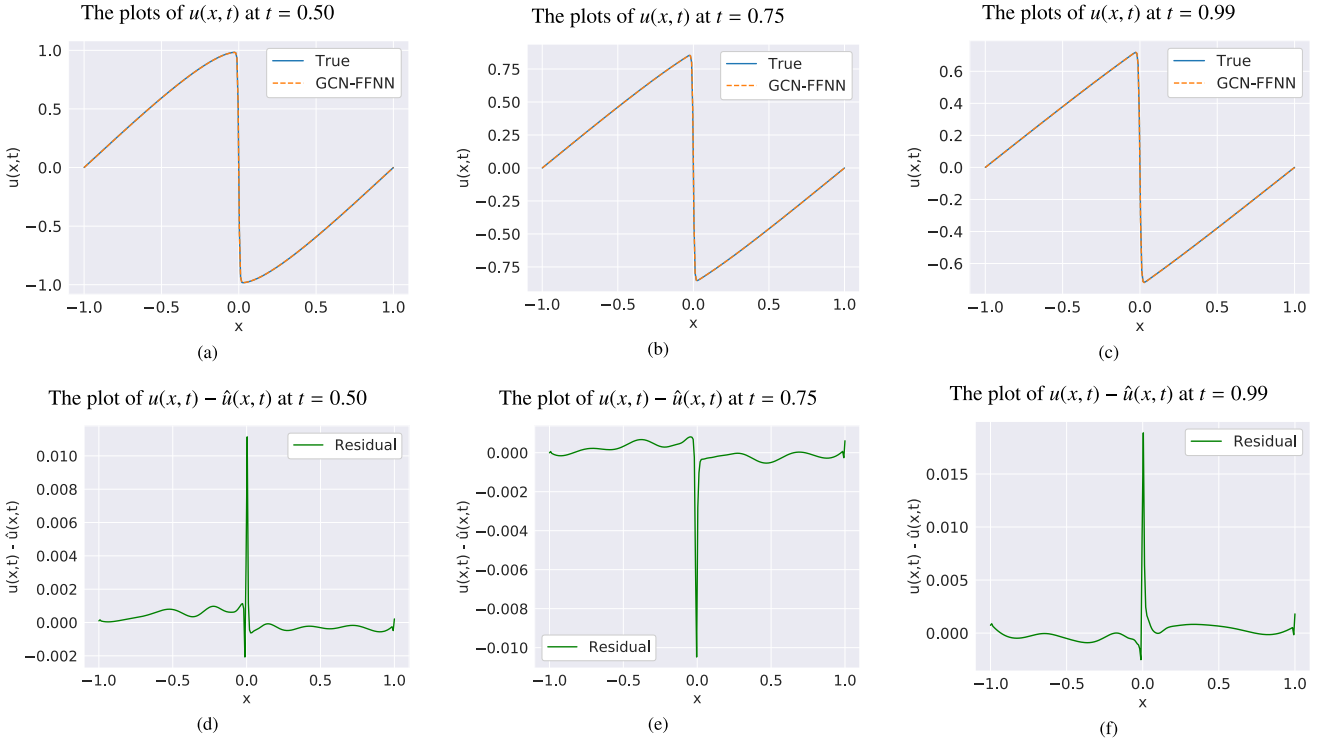

Fig. 6. Second scenario: The plots for the 1D-Burgers equation obtained by GCN-FFNN model corresponding to the second scenario where test nodes are from outside the domain. (a), (b) and (c): The true and approximate solution obtained by GCN-FFNN model. (d), (e) and (f): The obtained residuals $u(x,t) - \hat{u}(x,t)$. The data at $t = 0.50$ and $t = 0.75$ belong to the training set, while the data at $t = 0.99$ is from test set.

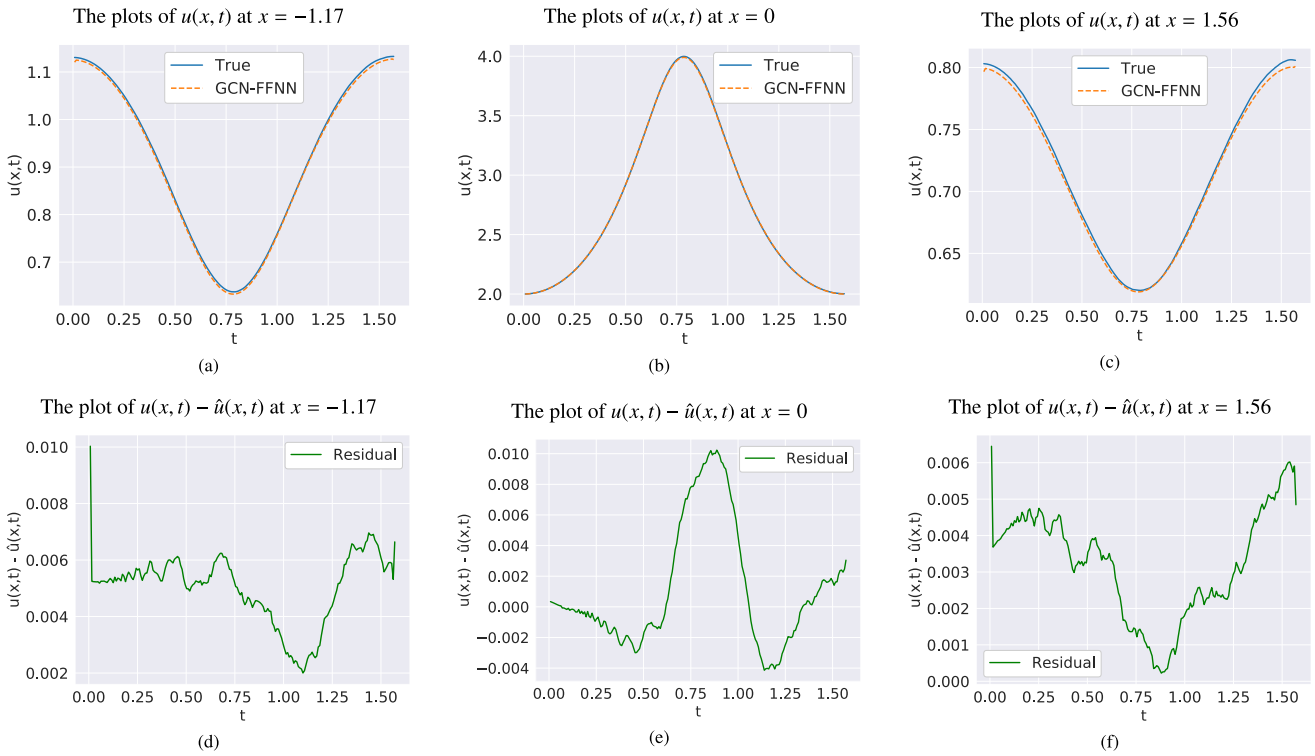

Fig. 7. First scenario: The plots for the 1D-Schrödinger equation obtained by GCN-FFNN model corresponding to the first scenario where test nodes are from inside the domain. (a), (b) and (c): The true and approximate solution obtained by GCN-FFNN model. (d), (e) and (f): The obtained residuals $u(x,t) - \hat{u}(x,t)$. The data at $x = -1.17$ belongs to the training set, while the data at $x = 0$ and $x = 1.56$ are from test set.

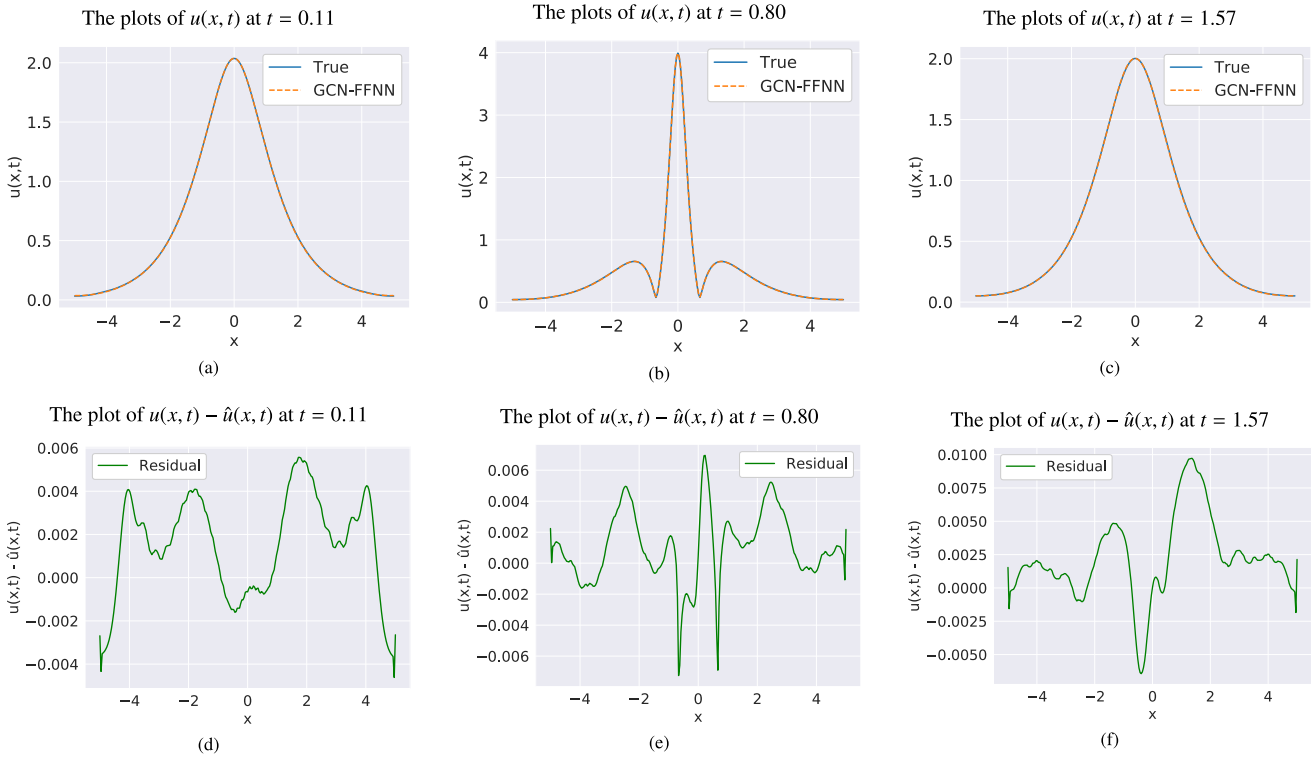

Fig. 8. Second scenario: The plots for the 1D-Schrödinger equation obtained by GCN-FFNN model corresponding to the second scenario where test nodes are from outside the domain. (a), (b) and (c): The true and approximate solution obtained by GCN-FFNN model. (d), (e) and (f): The obtained residuals $u(x,t) - \hat{u}(x,t)$. The data at $t = 0.11$ and $t = 0.80$ belong to the training set, while the data at $t = 1.57$ is from test set.

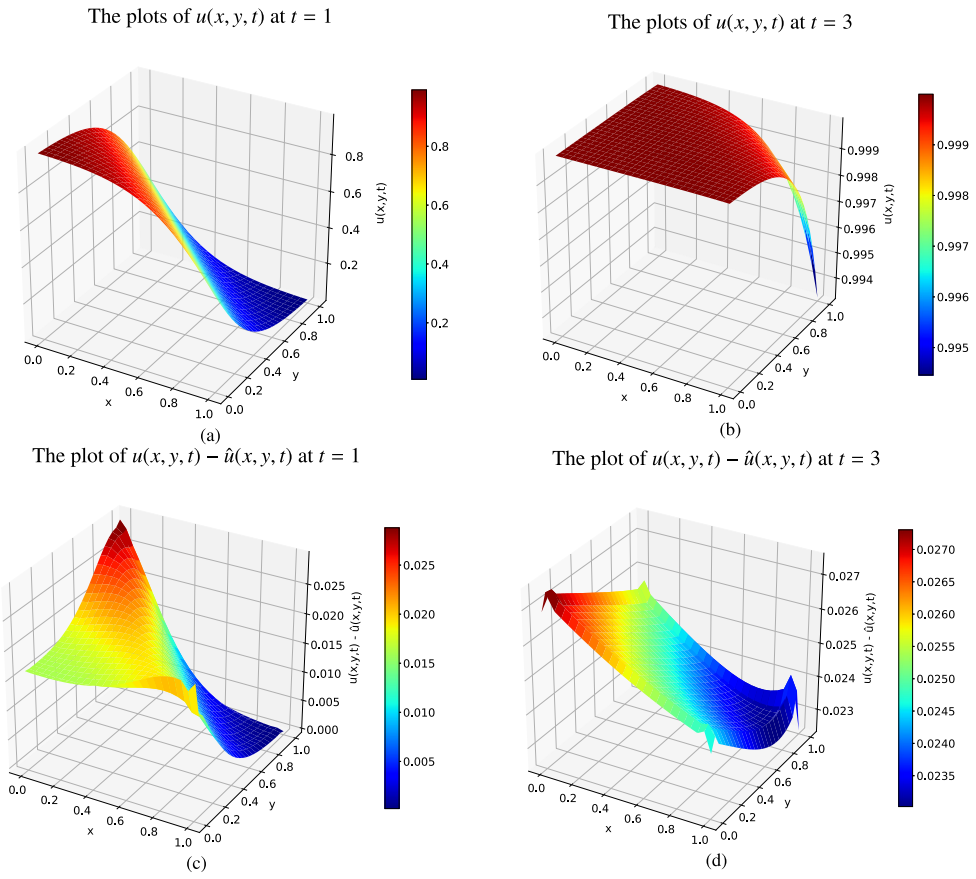

Fig. 9. The plots for the 2D-Burgers equation obtained by GCN-FFNN model. (a) and (b): The true solution $u(x, y, t)$. (c) and (d): The obtained residuals $u(x, y, t) - \hat{u}(x, y, t)$. The data at $t = 1$ and $t = 3$ are from the training set and test set (outside the domain), respectively.

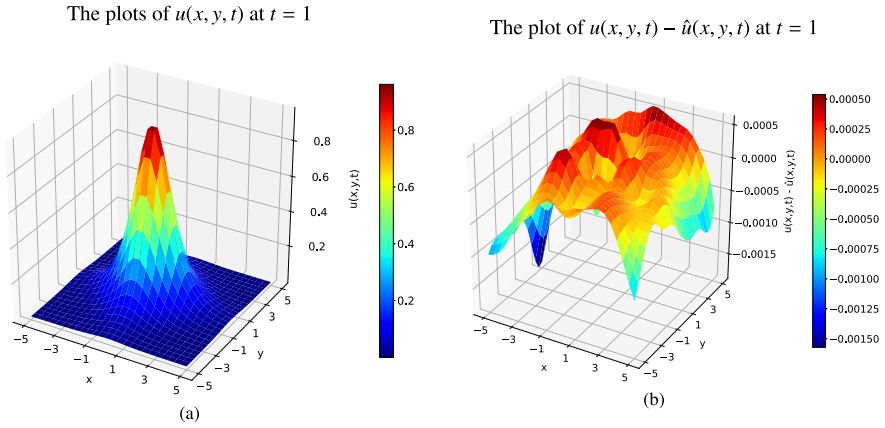

Fig. 10. The plots for the 2D-Schrödinger equation obtained by GCN-FFNN model for the outside domain test at $t = 1$. (a): The true solution $u(x, y, t)$. (b): The obtained residuals $u(x, y, t) - \hat{u}(x, y, t)$.

and the outside domain test nodes are selected from $0.9 < t \leqslant 1$. As can be seen from Table 3, for the 2D-Schrödinger equation, FFNN and GCN-FFNN models achieved comparable results on inside domain test nodes using both MSE and infinity norm metrics. For outside domain test nodes, the GCN-FFNN model outperforms other models using the MSE metric while it also achieved comparable results to the FFNN model in terms of infinity norm metric. The true solution and the obtained residual at outside domain test time $t = 1$ are shown in Fig. 10 (a) and (b), respectively.

4. Conclusion and future work

In this paper, a new two-stream architecture based on graph convolutional network (GCN) and feed-forward neural networks (FFNN) is developed for solving partial differential equations (PDEs). The model learns from both grid and graph input representations obtained by discretizing the domain of the given PDE. The proposed model is examined on four nonlinear PDEs, i.e. 1D-Burgers, 1D-Schrödinger, 2D-Burgers and 2D-Schrödinger equa-

tion. The performance of the models are evaluated on test data located inside and outside the domain. Thanks to the incorporation of both types of input representations, the proposed GCN-FFNN model often outperforms the other tested models for the studied PDEs. The proposed GCN-FFNN model can also be viewed as one of the first attempts in integrating the multi-modality or multi-view deep learning approaches in learning the solution of differential equations. The proposed model allows to exploit complementary information of multiple features or modalities in the context of new representation learning of PDE solutions. For more practical applications, the proposed model can potentially be integrated in simulation software that are used to simulate PDEs arising in real world applications such as astronautics, biomechanics, chemical and mechanical engineering, fluid mechanics and geophysical flows. In this work, we trained the models with L-BFGS in a full-batch approach. However, with an increasing number of nodes, node attributes and/or connected edges, the size of the graph increases which can potentially prevent a full batch approach for training the networks due to memory constraints. This also limits a much finer discretization of the domain. For future work, one may consider training the model in batches with different optimization methods, which would also allow a much finer discretization. Additionally, the number of edges in the graph can be increased with the second or third neighborhood, which can allow GCN to better incorporate the neighborhood information. The implementation of our GCN-FFNN model is available at [2].

CRediT authorship contribution statement

Onur Bilgin: Methodology, Software, Validation, Investigation. **Thomas Vergutz:** Methodology, Software, Validation, Investigation. **Siamak Mehrkanoon:** Methodology, Conceptualization, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. Mohanty, An unconditionally stable finite difference formula for a linear second order one space dimensional hyperbolic equation with variable coefficients, Appl. Math. Comput. 165 (1) (2005) 229–236.
[2] V. Thomée, From finite differences to finite elements a short history of numerical analysis of partial differential equations, Numerical analysis, in: Historical developments in the 20th century, 2001, pp. 361–414.
[3] A.A. Abushama, B. Bialecki, Modified nodal cubic spline collocation for poisson's equation, SIAM J. Numer. Anal. 46 (1) (2008) 397–418.
[4] M. Kumar, Y. Gupta, Methods for solving singular boundary value problems using splines: a review, J. Appl. Math. Comput. 32 (1) (2010) 265–278.
[5] F. Shakeri, M. Dehghan, A finite volume spectral element method for solving magnetohydrodynamic (mhd) equations, Appl. Numer. Math. 61 (1) (2011) 1–23.
[6] A. Taleei, M. Dehghan, Time-splitting pseudo-spectral domain decomposition method for the soliton solutions of the one-and multi-dimensional nonlinear schrödinger equations, Comput. Phys. Commun. 185 (6) (2014) 1515–1528.
[7] S. Mehrkanoon, J.A. Suykens, Learning solutions to partial differential equations using ls-svm, Neurocomputing 159 (2015) 105–116.
[8] A.J. Meade Jr, A.A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Modell. 19 (12) (1994) 1–25.
[9] H. Lee, I.S. Kang, Neural algorithm for solving differential equations, J. Comput. Phys. 91 (1) (1990) 110–131.
[10] B.P. van Milligen, V. Tribaldos, J. Jiménez, Neural network differential equation and plasma equilibrium solver, Phys. Rev. Lett. 75 (20) (1995) 3594.
[11] P. Ramuhalli, L. Udpa, S.S. Udpa, Finite-element neural networks for solving differential equations, IEEE Trans. Neural Networks 16 (6) (2005) 1381–1392.
[12] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Networks 9 (5) (1998) 987–1000.
[13] B. Choi, J.-H. Lee, Comparison of generalization ability on solving differential equations using backpropagation and reformulated radial basis function networks, Neurocomputing 73 (1–3) (2009) 115–118.
[14] Y. Shirvany, M. Hayati, R. Moradian, Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations, Appl. Soft Comput. 9 (1) (2009) 20–29.
[15] S. Mehrkanoon, T. Falck, J.A. Suykens, Approximate solutions to ordinary differential equations using least squares support vector machines, IEEE Trans. Neural Networks Learn. Syst. 23 (9) (2012) 1356–1367.
[16] S. Mehrkanoon, J.A. Suykens, LS-SVM approximate solution to linear time varying descriptor systems, Automatica 48 (10) (2012) 2502–2511.
[17] S. Mehrkanoon, J.A. Suykens, LS-SVM based solution for delay differential equations, in: Journal of Physics: Conference Series, Vol. 410, IOP Publishing, 2013, p. 012041.
[18] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Networks 11 (5) (2000) 1041–1049.
[19] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561.
[20] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[21] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.
[22] B. Yu, et al., The deep ritz method: a deep learning-based numerical algorithm for solving variational problems, arXiv preprint arXiv:1710.00211.
[23] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, Q. Yang, Large-scale hierarchical text classification with recursively regularized deep graph-cnn, in: Proceedings of the 2018 world wide web conference, 2018, pp. 1063–1072.
[24] N. Zhang, S. Deng, Z. Sun, G. Wang, X. Chen, W. Zhang, H. Chen, Long-tail relation extraction via knowledge graph embeddings and graph convolution networks, arXiv preprint arXiv:1903.01306.
[25] J. Johnson, A. Gupta, L. Fei-Fei, Image generation from scene graphs, in, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 1219–1228.
[26] O. Litany, A. Bronstein, M. Bronstein, A. Makadia, Deformable shape completion with graph convolutional autoencoders, in, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 1886–1895.
[27] T. Stańczyk, S. Mehrkanoon, Deep graph convolutional networks for wind speed prediction, in: Proc. of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN, 2021, pp. 147–152.
[28] D. Aykas, S. Mehrkanoon, Multistream graph attention networks for wind speed forecasting, in, in: IEEE Symposium Series on Computational Intelligence (SSCI) IEEE, 2021, pp. 1–8.
[29] Y. Li, F. Mei, Deep learning-based method coupled with small sample learning for solving partial differential equations, Multimedia Tools Appl. 80 (11) (2021) 17391–17413.
[30] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907.
[31] M. Poli, S. Massaroli, J. Park, A. Yamashita, H. Asama, J. Park, Graph neural ordinary differential equations, arXiv preprint arXiv:1911.07532.
[32] D.C. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Math. Programm. 45 (1) (1989) 503–528.

Onur Bilgin received the Dipl.-Ing. degree in Mechanical Engineering from RWTH (Aachen, Germany) in 2012. After completing his Dipl.-Ing., he worked from 2013 to 2020 as a Design Engineer in Dürr Ecoclean GmbH (Monschau, Germany). In 2022, he received the M.Sc. degree in Data Science for Decision Making from Maastricht University (The Netherlands). He is currently a Graduate Student at the University of South Florida (Tampa, USA). His current research interests are deep learning, natural language processing, and computational science.

[2] https://github.com/onurbil/pde-gcn

Thomas Vergutz received a B.Sc degree in Mechatronics Engineering from Universidade Federal de Uberlândia in 2013. He was from 2010 to 2013 the leader of a team in the robotics group (EDROM) at Universidade Federal de Uberlândia, this group won several awards and competitions during this time. He worked at AmBev, from 2014 to 2018 as an Automation Engineer and from 2018 on as an Artificial Intelligence Coordinator. In 2019 AmBev granted him a scholarship to pursue an M.Sc. He is currently a student at Maastricht University, pursuing an M.Sc. in Data Science for Decision Making.

Siamak Mehrkanoon received the B.Sc. degree in pure mathematics and the M.Sc. degree in applied mathematics from Iran University of Science and Technology, Tehran, Iran, in 2005 and 2007, respectively. He obtained the Ph.D. degrees in numerical analysis and machine learning from Universiti Putra Malaysia, Seri Kembangan, Malaysia, and Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 2011 and 2015, respectively. He was a Visiting Researcher with the Department of Automation, Tsinghua University, Beijing, China, in 2014, a Post-Doctoral Research Fellow with the University of Waterloo, Waterloo, ON, Canada, from 2015 to 2016, and a Visiting Post-Doctoral Researcher with the Cognitive Systems Laboratory, University of Tübingen, Tübingen, Germany, in 2016. He was an FWO Post-Doctoral Research Fellow with the Stadius Center for Dynamical Systems, Signal Processing and Data Analytics, KU Leuven from 2016 to 2018 and an Assistant Professor at the Department of Data Science and Knowledge Engineering (DKE), Maastricht University, the Netherlands from 2018-2022. He is currently an Assistant Professor at the Department of Information and Computing Sciences, Utrecht University, Utrecht, Netherlands and affiliated with Maastricht University as a free researcher. His current research interests include deep learning, neural networks, kernel-based models, numerical algorithms, optimization and computational science. He has been awarded several Grants including PDM from KU Leuven and prestigious Fund for Scientific Research from FWO Flanders.