Application of Deep Learning for Solving Partial Differential Equations and Developing Fluid Flow Solver

Jaydutt Kulkarni, Satbir Singh

[a]*Department of Mechanical Engineering*
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, 15213, PA, USA

Abstract

Partial Differential Equations (PDEs) are common in many engineering problems. Traditionally, PDEs are solved using numerical methods such as finite difference, finite volume, and finite element. Recent learning methods using neural network have been used to directly solve PDEs without the use of training data. In this work, the use of deep learning for solving several PDEs spanning from one-dimensional transient advection equation to two-dimensional transient advection-diffusion equation is explored. It is found that deep learning methods can provide certain advantages over traditional numerical methods. However, much progress is needed to develop robust and generalizable solvers. For example, neural network solutions to one-dimensional transient advection equation as well as two-dimensional steady state advection-diffusion equation are better than the traditional numerical method, and the neural network architecture was obtained with ease. But, as dimensionality of the given problem increases, deep learning solver requires significant user intervention to obtain a plausible solution.

Keywords: Predictive modelling, Physics Informed Neural Networks (PINNs), neural network, Computational Fluid Dynamics (CFD), Deep Learning, Partial Differential Equations (PDEs), numerical methods, fluid flow

1. Introduction

The advent of Machine Learning/Deep Learning and data analytics in scientific fields has led to finding alternative routes to solve problems with known solutions [1] or even discovering solutions to previously unsolved problems [2]. PDEs have many important uses in many fields including physics and engineering. Due to their importance and the fact that exact solutions sometimes do not exist and/or are computationally complex/expensive to evaluate, there is a need to approximate these PDEs as accurately as possible. The aim here is to use deep learning and obtain solutions for PDEs. It is posited that there are certain advantages of using deep learning methods over traditional numerical schemes, as will be demonstrated in the results. The advantages are including but not limited to a lesser dependency on grid size/type, thus allowing deep learning solvers to do better than numerical solvers.

Among many problems in the field of fluid mechanics, building a solver for the Navier Stokes equation in higher dimensions is one of the most challenging problems. In order to approach a solution to such a problem, it is prudent to perhaps address a simplified version of the physical phenomenon, which in this case is a transient convection diffusion transport equation. The task of training a deep learning algorithm to accurately mimic a non linear function in higher dimensions based on a few input and output data pairs seems naive at best. Fortunately, there are many cases that have already been tackled by ground-breaking work in the area of Physics Informed Neural Networks (PINNs) by Raissi et al. [3, 4, 5]. The primary work [3] has handled continuous time models such as Schrodinger equation (periodic boundary conditions, complex-valued solutions, as well as different types of non-linearities in the governing PDEs), and discrete time models such as Allen-Cahn equation. Later articles have demonstrated tackling Burgers' equation, the Korteweg–De Vries (KdV) equation, and non-linear Schrodinger's equation in two dimensions (t, x). There are deep learning solutions to semi-parabolic PDEs as well, as demonstrated by Han et al. [6], using Backward Stochastic Differential Equation. However, as can be seen in all these prior works, they are only able to handle transient PDEs if the governing equation exists in only one other dimension (i.e., x). Either that or they are able to handle steady state PDEs in two dimensions. In this article, the understanding of applying the same principles of solving PDEs in lower dimensions to PDEs governing higher dimensions is explored. The overarching idea is to make a generalizable solver based on an approach of encoding structured information into a learning algorithm, which has already been explored. The solution generated from such algorithms must be comparable to the approximations of traditional numerical methods. This comparison will also help set a performance benchmark when an analytical solution is not available.

Apart from this, it is also vital to understand why a deep learning approach might not lead to the desired solution. Krishnapriyan et al. [7] have already demonstrated a technique of how to visualize the multi-dimensional loss terrain that the optimizer of a neural network actually sees, albeit in simplified three dimensional space. This technique of using the two most dominant eigenvectors has been described by Yao et al. [8], and this approach has been leveraged in this article to understand why a deep learning solution might not be feasible for certain problems. Even though it is not a direct and emphatic proof of infeasibility, the loss landscape definitely helps us understand if a particular problem is too large or too

complex in the parameter-space, which causes the optimizer to get stuck at either a saddle point or a local minima. Since the number of weights are large, the space is n-dimensional (where n is the number of weights of the neural network) which means that the likeliest possibility is a saddle point. Such analysis would be very handy to quickly identify if a particular PDE can be solved using the PINN approach.

In this article, a way of solving PDEs that rely on the function approximation capabilities of neural networks and results in the construction of a solution is analyzed. This involves a feed-forward neural network as the basic approximation element, whose parameters (weights and biases) are adjusted to minimize an error function chosen on the basis of the task (in most cases, regression). To train the network, optimization techniques are employed, which in turn require the computation of the gradient of the error with respect to the network parameters. In the proposed approach the model function is expressed as the sum of multiple terms: the first term satisfies the initial/boundary conditions (depending on dimensionality of the problem) and has no trainable parameters. The second term involves a feed-forward neural network to be trained to satisfy the given PDE at all collocation points, as well as at edge cases. Since it is known that a multi-layer perceptron (MLP) can approximate any function to arbitrary accuracy [9], its properties can be leveraged and this kind of network architecture can be considered for solving various PDEs.

2. Problem Setup

The capability of deep neural networks as universal function approximators [10] has been leveraged in this work. This helps us handle the non-linearity of the problem without the use of linearization, or local time stepping. The developments in automatic differentiation [11] have been used to differentiate neural network w.r.t its input dimensions in order to approximate the PDE involved. This will allow us to approximate the PDE subject to certain constraints. The groundwork for this new paradigm in modelling was laid as previously discussed [3]. In order to apply these new techniques on pre-existing problems in the domain of fluid mechanics, different kinds of equations and conditions that will highlight the importance, use, and advantages of PINNs over more traditional methods, have been used. A combination of OpenFOAM, Python, and MATLAB has been used to generate the numerical scheme solutions.

2.1. 1D transient advection

A long, uniform Cartesian grid in x-direction is employed for each time interval Δt, as shown in *Fig. 1.* The 1D transient advection equation for scalar transport is a hyperbolic PDE and involves the temporal dimension, t, and a spatial dimension, x, and can be represented as follows:

$$\frac{\partial \phi}{\partial t} + u_x \frac{\partial \phi}{\partial x} = 0, \quad x \in \Omega \times [0, T], \tag{1}$$

$$\phi(0, x) = \phi_0(x), \quad x \in \Omega \times [0, T], \tag{2}$$

$$a(t, x)\phi + b(t, x)\frac{\partial \phi}{\partial \eta} = F(t, x) \quad x \in \partial\Omega, \ \ t \in [0, T], \tag{3}$$

where $\Omega \subset \Re$ is a one-dimensional domain ([0,400]), $(0, T]$ is the time interval, and ϕ_0 and F are the initial and boundary conditions respectively. The coefficients a and b define the boundary condition as one of the Dirichlet, Neumann, and Robin type boundary conditions [12]. It is constrained by certain initial conditions (I.C.) at time, t = 0 as well as boundary conditions (B.C.) at the extreme values of the x-dimension. The flow has uniform velocity in x-direction with uniform viscosity throughout the medium. The initial condition is given below:

$$\phi_0(x) = \begin{cases} 0 & x < 50 \\ 100 \sin \frac{\pi(x-50)}{60} & 50 \leq x < 110 \\ 0 & x \geq 110 \end{cases} \tag{4}$$

The boundary conditions applied in this particular problem are Dirichlet boundary conditions. $F(t, x) = 0$ at all boundary nodes. u_x is the convection velocity in the x-direction. Since this is purely an advection problem, there is no viscosity component to be considered. The convection velocity is initially fixed at $u_x = 0.8$, which is then varied in order to contrast and compare finite different methods with neural network solutions and their sensitivity to grid size and velocity. There is no data provided to the neural network for any of the collocation points in this domain. The only constraint for these collocation points is that the PDE evaluated has to equal zero.

2.2. 2D Steady State advection-diffusion

A uniform Cartesian grid with 561 nodes in both x- and y-directions is employed for the first half of this experiment. The second half is set up with a non-orthogonal triangular mesh with 2.5 x 10^7 nodes. This experiment is run on both orthogonal as well as non-orthogonal grids, as shown in *Fig. 2*, to understand the impact of grid orthogonality on PINN

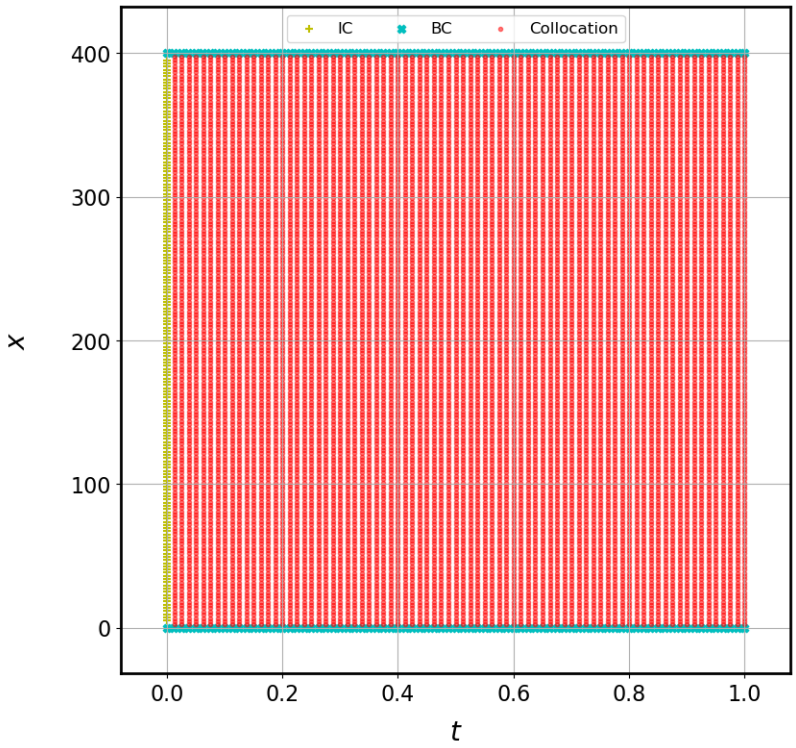

Figure 1: Orthogonal grid in t-x domain for the 1D transient advection problem showing initial condition nodes, boundary condition nodes with Dirichlet boundary conditions, and collocation nodes

solution compared to numerical schemes. Since this is now a steady state problem, there is no temporal dimension to consider. The PDE for this advection-diffusion phenomenon can be formulated as:

$$u_x \frac{\partial \phi}{\partial x} + u_y \frac{\partial \phi}{\partial y} = \nu \left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \right), \quad (x, y) \in \Omega \tag{5}$$

$$a(x, y)\phi + b(x, y)\frac{\partial \phi}{\partial \eta} = F(x, y) \quad (x, y) \in \partial\Omega, \tag{6}$$

where $\Omega \subset \Re^2$ is a rectangular domain ([0,3.5] x [0,3.5]), and F is the boundary condition. The B.C.s applied in this problem are Dirichlet as well as Neumann B.C.s at the extreme values of the x- and y-dimensions. The flow has uniform velocity in both x and y directions with uniform viscosity throughout the medium. The boundary conditions are given below:

$$\phi_{x=0} = 200, \quad \phi_{y=0} = 0, \qquad \text{(Dirichlet B.C.s)} \tag{7}$$

$$\left.\frac{\partial \phi}{\partial \eta}\right|_{x=x_{max}} = 0, \quad \left.\frac{\partial \phi}{\partial \eta}\right|_{y=y_{max}} = 0, \qquad \text{(Neumann B.C.s)} \tag{8}$$

The viscosity is initialized at $\nu = 0.01$. This is then lowered in order to obtain higher Reynolds numbers in order to test the robustness of neural network architecture. For a higher Reynolds number, the numerical method almost certainly breaks down ($Re = 2\text{x}10^7$), so the idea is to compare performance of various numerical schemes at such high Reynolds number with the neural network solution. Again, there is no data provided to the neural network for any of the collocation points in this domain. The only constraint for these collocation points is that the PDE evaluated has to equal zero, and that the Neumann B.C.s are satisfied via a loss function which evaluates the difference between the scalar values of the two adjacent rows of nodes at each boundary.

2.3. 2D transient advection-diffusion

A uniform Cartesian grid with 80 cells in both x- and y-directions is employed with the temporal dimension divided into 200 cells, as shown in *Fig. 3*. The 2D transient advection-diffusion equation for scalar transport is studied, which involves three dimensions - t, x, and y and can be written as follows:

$$\frac{\partial \phi}{\partial t} + u_x \frac{\partial \phi}{\partial x} + u_y \frac{\partial \phi}{\partial y} = \nu^2 \left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \right), \quad (x, y) \in \Omega \times [0, T], \tag{9}$$

$$\phi(0, x, y) = \phi_0(x, y), \quad (x, y) \in \Omega \times [0, T], \tag{10}$$

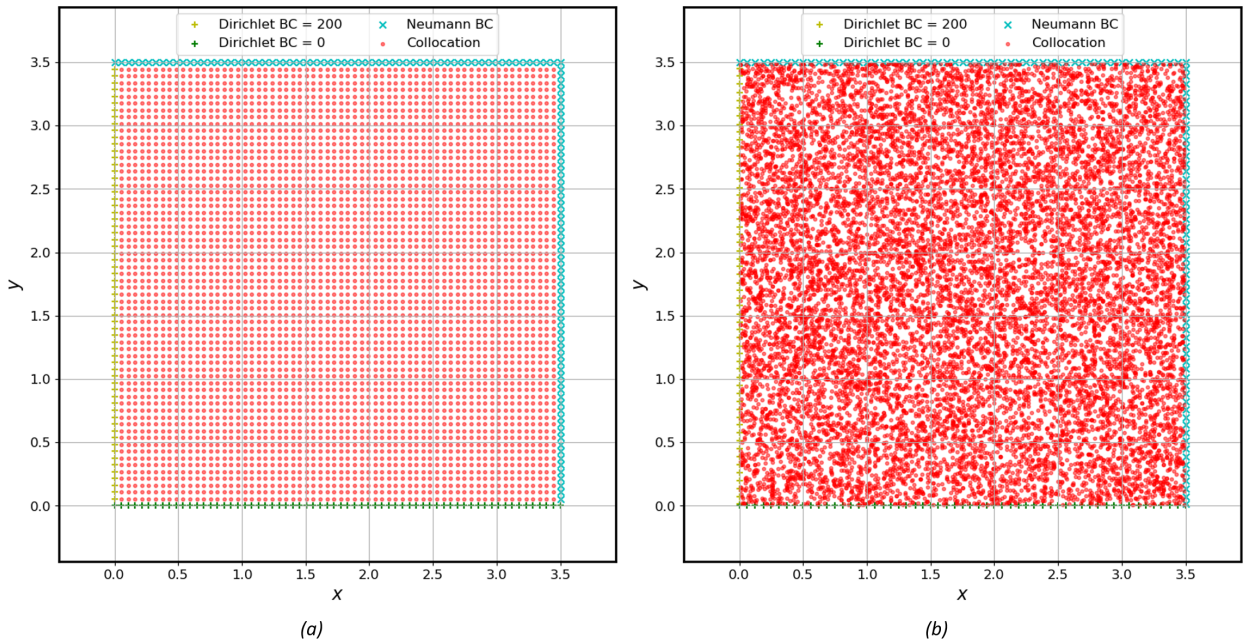

Figure 2: (a) Orthogonal grid in x-y domain and (b) Non-orthogonal grid in x-y domain using triangular (tetrahedral in 2D) elements, for the 2D steady state advection-diffusion problem showing boundary condition nodes with Dirichlet as well as Neumann boundary conditions, and collocation nodes

$$a(t,x,y)\phi + b(t,x,y)\frac{\partial \phi}{\partial \eta} = F(t,x,y) \quad (x,y) \in \partial\Omega, \;\; t \in (0,T], \tag{11}$$

where $\Omega \subset \Re^2$ is a rectangular domain ([0,2] x [0,2]), $(0,T]$ is the time interval, and ϕ_0 and F are the initial and boundary conditions respectively. It is constrained by certain I.C.s at time, t = 0 as well as B.C.s at the extreme values of the x- and y-dimensions. The flow has uniform velocity in both x and y directions with uniform viscosity throughout the medium. The initial condition is given below:

$$\phi_0(x,y) = \exp\left(-\frac{(x-0.5)^2}{\nu} - \frac{(y-0.5)^2}{\nu}\right) \tag{12}$$

The analytical solution in Eq. (13) is the boundary condition applied to this problem, which is the transient advection-diffusion of a Gaussian pulse. The boundary conditions applied in this problem as well are Dirichlet boundary conditions. $u_{x(y)}$ is the convection velocity in the x(y)-direction and ν is the viscosity. The viscosity is fixed as $\nu = 0.01$ unless there are experiments done in order to understand the effect of viscosity on the accuracy of the neural network solution. The default Reynolds number is chosen to be 2 by setting the convection velocities, $u_x = u_y = 0.8$. The other Reynolds number used for experiments was 200 by setting $u_x = u_y = 80$. Yet again, there is no data provided to the neural network for any of the collocation points in this domain. The only constraint for these collocation points is that the PDE evaluated has to equal zero.

$$\phi_{ana}(t,x,y) = \frac{1}{4t+1}\exp\left(-\frac{(x-tu_x-0.5)^2}{\nu(4t+1)} - \frac{(y-tu_y-0.5)^2}{\nu(4t+1)}\right) \tag{13}$$

3. Deep Learning Model

As discussed earlier, neural network as a universal function approximator, can be used to deal with arbitrary regression problems. Compared with the general multiple linear regression model, neural network hierarchically adds hidden layers to represents the hidden feature level and adds an activation function at the output of each hidden layer to increase the non-linearity of the model. A typical neural network training process should include two steps, namely forward propagation and backward propagation. First, in forward propagation, neural network multiplies the input data by the existing weight and calculates the output of the first layer through the activation function, and then uses the output of this hidden layer as the input of the next hidden layer. When this weighted input goes through all hidden layers, the predicted result is derived after output layer. After obtaining the output result, loss function is used to compare the difference between the predicted value and the ground truth. The second step is back propagation, that is, use the chain rule to calculate the derivative of the loss function value to each weight, and then use the gradient based optimization algorithm to update the weight value.

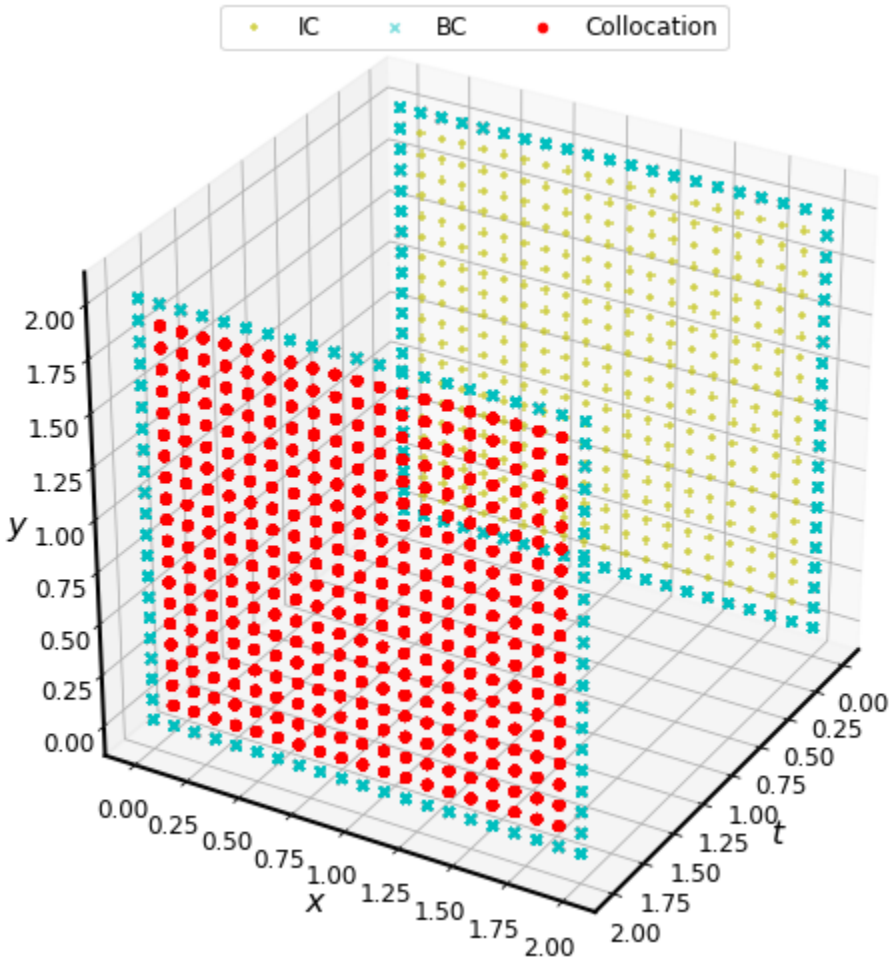

Figure 3: Orthogonal grid in t-x-y domain for the 2D transient advection-diffusion problem showing initial condition nodes, boundary condition nodes with Dirichlet boundary conditions, and collocation nodes at two separate temporal instances, t = 0 and t = 2

In training, a forward propagation and a back propagation together is called an epoch. When using the trained model to predict, only forward propagation needs to be performed to get the prediction, which is called the "evaluation" mode of the model [13].

This article utilizes MLPs to construct a deep neural network (DNN). An MLP is a fully connected class of feed-forward artificial neural network (ANN) and refers to networks composed of multiple layers of perceptrons with threshold activation. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. As explained above, learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. This is an example of supervised learning, and is carried out through back-propagation, a generalization of the least mean squares algorithm in the linear perceptron. This ability of MLPs to solve problems stochastically is very useful for obtaining solutions to complex problems such as solutions to PDEs.

3.1. Input Data

One of the main advantages of leveraging DNN for solving PDEs is that the actual input data to be provided is small compared to the total amount of data being evaluated. The only data points in the domain for which the governing function is evaluated and provided to the DNN as input are the initial condition points (wherever applicable) and the boundary condition points. For all the other points in the domain, referred to as collocation points earlier in the article, the only constraint imposed is the PDE itself. Parameterized and non-linear PDEs can be written down in a general form as:

$$\phi_t + \mathcal{N}[\phi; \lambda] = 0, \quad (x, y) \in \Omega, \;\; t \in [0, T], \tag{14}$$

where $\phi(t, x, y)$ denotes latent solution, $\mathcal{N}[\cdot; \lambda]$ is a non-linear operator parameterized by λ, and $\Omega \subset \Re^2$. Taking an example of the 2D transient advection-diffusion equation, this corresponds to the case where $\mathcal{N}[\phi; \lambda] = \lambda_1 \phi_x + \lambda_2 \phi_y - \lambda_3 \phi_{xx} - \lambda_4 \phi_{yy}$ and $\lambda = (\lambda_1, \lambda_2, \lambda_3, \lambda_4)$. Here, the subscripts denote partial differentiation in either time or space. The

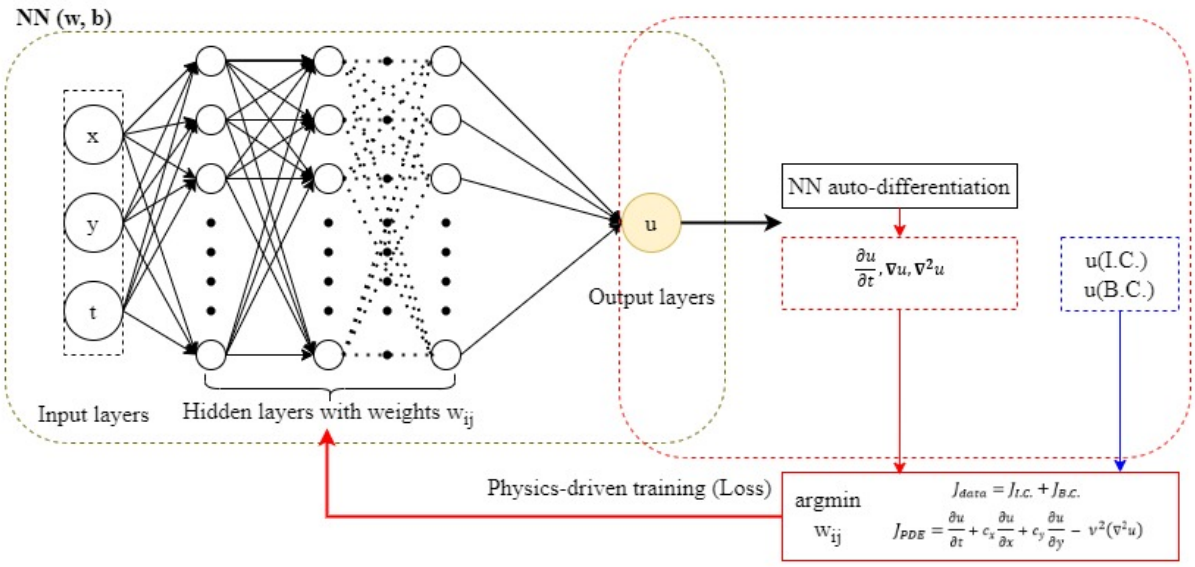

Figure 4: Network architecture and workflow of the PINN model used to predict the value of scalar transport at any point in the given domain. This network utilizes 8 layers with 20 neurons in each layer

problem statement thus becomes: *"given fixed model parameters λ how can the unknown hidden state u(t,x,y) of the system be computed ?"* The Eq. (14) can be rewritten as follows:

$$f(t, x, y) = \phi_t + \mathcal{N}[\phi], \tag{15}$$

where $\phi(t, x, y)$ can be approximated using a DNN. This formulation for f(t,x,y) is essentially PINN. This network can be derived by applying the chain rule for differentiating compositions of functions using automatic differentiation [11]. It also has the same parameters as the network representing $\phi(t, x, y)$, but with different activation functions due to the action of the differential operator, $\mathcal{N}$ [5]. The nature of the PDE can be learned by minimizing the mean squared error loss, which is explained in *Section 3.2.*

3.2. Architecture

The DNN model in this article utilises an MLP setup which has fully connected layers, where each node in one layer connects with a certain weight w_{ij} to every node in the following layer. An illustration of this can be seen in *Fig. 4*. The input layer consists of all the data points in the system, and its dimensions are equal to the number of dimensions of the problem. These independent variables are fed to the DNN, and the expected output is the dependent variable to be predicted, i.e., ϕ. DNN consisting of MLPs are widely used in non-linear regression, linearly-inseparable classification tasks, as well as tabular datasets and they exhibit good performance [14]. Before being passed to the first linear layer, the input goes through a scaling layer which normalizes input between -1 and 1. This has shown to improve network performance [15]. After that, for every linear layer, the layer of neurons has a set of weights assigned to them (either arbitrarily or by design) which determines the importance of each signal coming into that layer.

For the purposes of these experiments, the Glorot uniform method of weight initialization [16] is used. The Glorot (or Xavier) initialization attempts to alleviate the problem of vanishing gradients by setting the initial weights as a distribution where the variance of the distribution is dependent on the number of input and output layers. Once the input is converted into a weighted sum by all neurons of that layer, it is then passed to a batch normalization layer (wherever applicable). Due to the number of layers being significant, batch-normalization [17] is used to avoid over-fitting and improve model performance. The next step is an activation function which determines the output of that layer based on the decided threshold. The activation functions used are either hyperbolic tan (tanh) or rectified linear units (ReLU). Tanh is centred around zero which is why it is preferred, however if there is an issue of vanishing gradients [18], ReLU can be used instead. This arrangement of linear layer - batch norm - activation is repeated as many times as the number of hidden layers. The final activation layer is connected to the output layer, which is another linear layer characterised by the number of dependent variables.

The parameters of the neural network described in Eq. (15) can be learned by minimizing the loss function. In all the cases, a Mean-Squared Error (MSE) loss function or the L^2-norm is used, which represents the scale of difference between prediction and ground truth. This loss function ($\mathcal{J}$) can be constructed as:

$$\mathcal{J} = \mathcal{J}_{data} + \mathcal{J}_{PDE}, \tag{16}$$

where

$$\mathcal{J}_{data} = \mathcal{J}_{I.C.} + \mathcal{J}_{B.C.} = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} |\phi(t^i_{data}, x^i_{data}, y^i_{data}) - \phi^i|^2, \tag{17}$$

and

$$\mathcal{J}_{PDE} = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t^i_f, x^i_f, y^i_f)|^2. \tag{18}$$

Here, $\{t^i_{data}, x^i_{data}, y^i_{data}, u^i_{data}\}_{i=1}^{N_{data}}$ denote the initial and boundary training data on $\phi(t, x, y)$ and $\{t^i_f, x^i_f, y^i_f\}_{i=1}^{N_f}$ specify the collocation points for $f(t, x, y)$. The loss $\mathcal{J}_{data}$ corresponds to the initial and boundary data while $\mathcal{J}_{PDE}$ enforces the structure imposed by Eq. (14) at a finite set of collocation points. In all cases pertaining to data-driven solution of partial differential equations, the total number of training data N_{data} is relatively small (a few hundred up to a few thousand points), compared to the number of collocation points, N_f. There are a lot of algorithms that can help find the minima of the loss function, such as stochastic gradient descent [19], mini-batch gradient descent [20], and adaptive moment estimation (ADAM) [21]. This article uses ADAM algorithm, which has all the advantages of gradient descent but also tries to prevent training loop from stalling at local minima or saddle point. Learning rate is another important hyperparameter that decides the speed, convergence, and step magnitude of training. A large learning rate which will skip past the solution or bounce back and forth between two points without converging is not desirable. On the other hand, a very small learning rate which would prevent the optimizer from climbing out of local minima is also unacceptable. For all cases, a learning rate of 0.01 has been used with a step size scheduler to adapt to the training algorithm as the DNN gets closer to the optimum. This means that after certain number of epochs, the learning value is reduced by a factor of $\gamma = 0.5$. Despite the fact that there is no theoretical guarantee that this procedure converges to a global minimum, the empirical evidence indicates that, if the given partial differential equation is well-posed and its solution is unique, PINN is capable of achieving good prediction accuracy given a sufficiently expressive neural network architecture and a sufficient number of collocation points, N_f. This observation results in the optimization landscape generated by MSE loss, and can be inspected closely in case of a lack of convergence.

4. Results and Discussion

The neural network is trained for all cases discussed in *Section 2*. In order to understand the benefits of using PINN, they are subjected to conditions where traditional numerical methods generally fail. These include conditions like high Courant number (grid size) [22], high Reynolds number, and non-orthogonal grids. The PINN solutions are compared to numerical solutions as well as closed form or analytical solutions to understand and contextualize their performance. The loss landscape can also be visualized for cases where PINN is unable to converge to the analytical solution in order to understand how the optimizer might be seeing the loss function domain in n-dimensions. This is done by perturbing the (trained) model across the first two dominant Hessian eigenvectors and computing the corresponding loss values [7]. In machine learning terms, Principal Component Analysis (PCA) [23] is being performed by selecting the top two vectors that explain most of the variance. This tends to be more informative than perturbing the model parameters in random directions [8, 24, 25], and can help us understand the reasons behind the lack of convergence for a particular case.

4.1. 1D Transient advection

The model is trained using initial and boundary condition nodes as input data, as well as the PDE residual. The grid size is quantified by Courant number (C), which is a dimensionless value representing the time a particle stays in one cell of the mesh, and is defined as:

$$C = \left| u \frac{\Delta t}{\Delta x} \right|, \tag{19}$$

where, u is the velocity magnitude, Δt is the time step size, and Δx is the length between mesh elements. The experiment was performed on 3 different Courant numbers, 0.45, 1.0, and 2.0. According to the Courant–Friedrichs–Lewy(CFL) condition [22], if the Courant number exceeds 1, the time step is too large to see the particle in one cell and it "skips" the cell. This leads to instability in the numerical method, causing it to blow up. The focus will be to demonstrate the advantage of PINN (grid size invariance) over certain numerical schemes. *Fig. 5* shows a comparative study of two different numerical schemes in comparison with the analytical and PINN solution for three different Courant numbers. The Courant numbers are chosen such that they lie on either side as well as on the CFL stability condition. Explicit numerical schemes in spatial dimension have not been considered for comparison since they are unconditionally unstable and can completely break down and display oscillations of large magnitude. On the other hand, implicit or upwind numerical schemes are unconditionally stable, although they might be subject to damping or numerical diffusion. Numerical diffusion is when the simulated medium exhibits a higher diffusivity than the true medium. This phenomenon can be particularly egregious when the system should

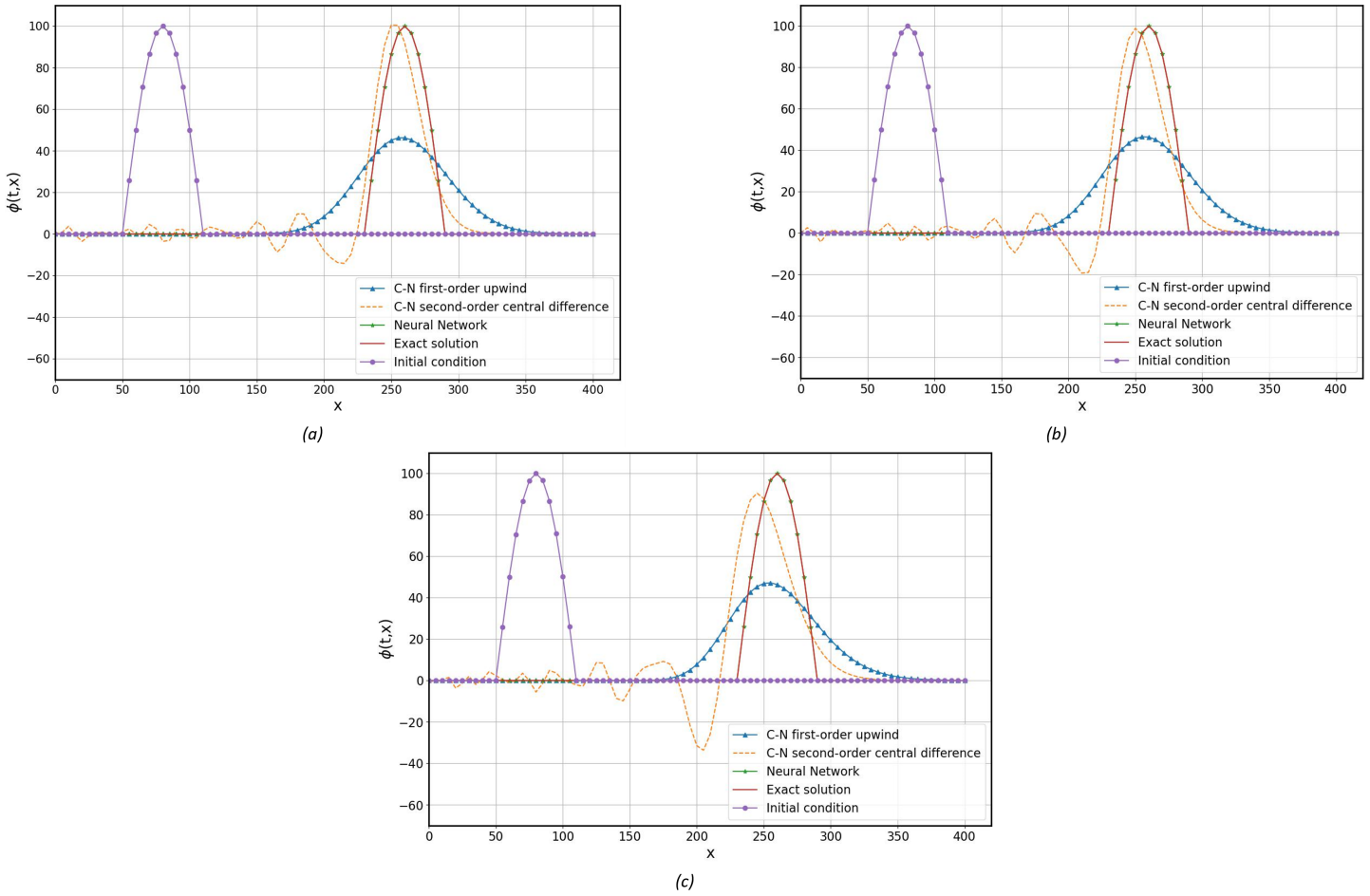

Figure 5: $\phi(t, x)$ vs x used for comparison between solutions of 1D transient advection equation obtained analytically, via C-N first-order upwind scheme, via C-N second-order central difference method, and using PINN, for (a) C = 0.45, (b) C = 1.0, and (c) C = 2.0

not be diffusive at all, as is the case during pure advection. Crank-Nicolson (C-N) first-order backward difference (in space) scheme [26] as well as C-N second-order central difference (in space) scheme [27] were chosen as the two numerical schemes for the purposes of this comparison.

As can be seen in *Fig. 5*, for all three Courant numbers, C-N second-order central difference scheme displays numerical oscillations. Numerical oscillations are undesirable artifacts in CFD simulations, and they occur when a central difference scheme is used to discretize fluid equations on a coarse grid. These spurious oscillations are comparatively damped till the CFL condition is satisfied and they increase in magnitude once C > 1. Also, the peak of the solution gets more out-of-phase with the analytical solution. On the other hand, the C-N first-order backward difference scheme overcomes these spurious oscillations, but exhibits numerical diffusion. The peak also gets off-centred w.r.t. the analytical solution as C increases. The PINN solution performs much better than both C-N schemes and mimics the analytical solution almost exactly. This shows the advantage of using PINN over certain numerical schemes which employ central/backward differencing to compute derivatives.

4.2. 2D Steady State advection-diffusion

The model is trained using all the training data, and for three different Reynolds numbers: 2, 200, and 2 x 10^7. This is to understand the effect of viscosity values on the solution and to observe if there are limitations to using numerical methods such as false diffusion or numerical oscillations for very high Reynolds numbers. The error in the upwind differencing scheme has a diffusion-like appearance in two- or three-dimensional co-ordinate systems for non-orthogonal grids and is referred as "false diffusion". False diffusion errors in numerical solutions of convection-diffusion problems, in two- and three-dimensions, arise from the numerical approximations of the convection term in the conservation equations. As the diffusivity of the system is reduced, the false diffusion component starts becoming relatively more prominent, which is when the importance of PINN can be highlighted. Also while dealing with the convection term, when the Reynolds number exceeds a critical value, it results in numerical oscillations. These spurious oscillations can also occur at sharp discontinuities due to the numerical scheme approximating with a larger order of magnitude of error near these areas. In order to demonstrate the advantage of PINN, a Reynolds number of Re = 2 x 10^7 will be used for all further results.

The 2D Steady State case was simulated in OpenFOAM using a number of numerical schemes such as Gauss linear interpolation (default scheme), Gauss cubic interpolation, Quadratic Upstream Interpolation for Convective Kinematics

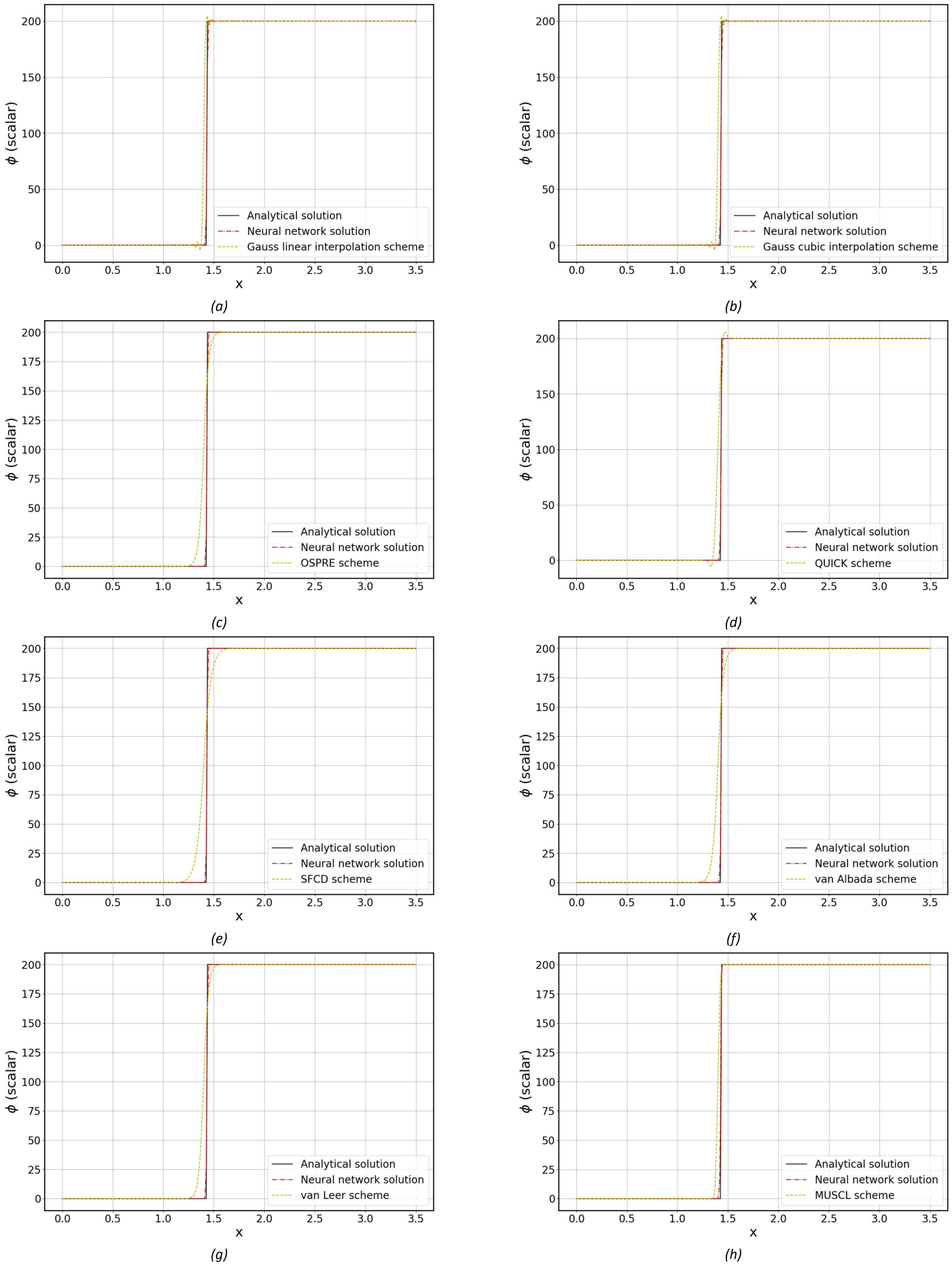

Figure 6: $\phi(x, y)$ vs x used for comparison between solutions of 2D steady state advection-diffusion equation obtained analytically, using PINN, and numerical schemes in OpenFOAM such as: (a) Gauss linear interpolation, (b) Gauss cubic interpolation, (c) OSPRE, (d) QUICK, (e) SFCD, (f) van Albada, (g) van Leer, and (h) MUSCL, at y = 1.4

(QUICK) [28], van Albada [29], van Leer [30], Self-filtered central differencing (SFCD) [31], Optimum Symmetric Polynomial Ratio Expression (OSPRE) [32], and Monotonic Upstream-centered Scheme for Conservation Laws (MUSCL) [33]. All these different numerical methods were compared with the analytical solution on an orthogonal grid for bench-marking purposes. Out of all these schemes, the analytical solution came closest to the PINN solution, as can be seen in *Fig. 6*. Once again, PINN outperforms all the numerical schemes utilized and displays no numerical artifacts. The sharp discontinuities, with which many numerical schemes fail to deal in an error-free manner, are handled in a much more accurate manner as well.

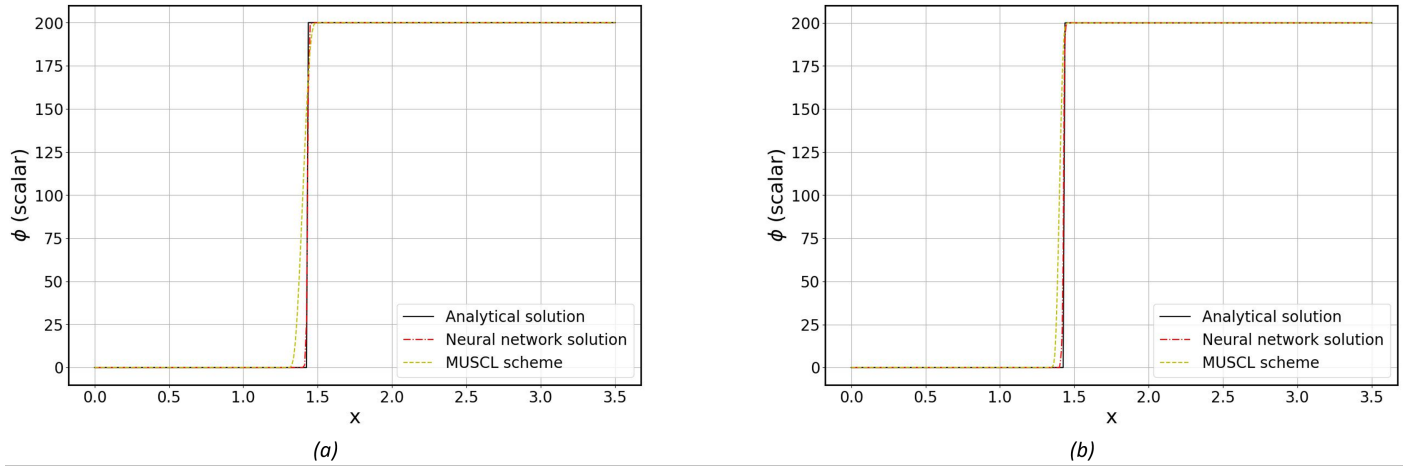

Figure 7: $\phi(x, y)$ vs x used for comparison between solutions to 2D steady state advection-diffusion equation obtained analytically, using PINN, and MUSCL numerical scheme at y = 1.4 on (a) a coarse orthogonal grid (with 281 nodes in both x- and y-directions), and (b) a finer orthogonal grid (with 561 nodes in both x- and y-directions)

Looking at the accuracy of predictions, presence of numerical artifacts, and stability of different schemes compared to the analytical solution, the MUSCL scheme is the best performing numerical scheme of the lot. In order to understand the impact of grid sparsity on the best performing numerical scheme as well as PINN, model performances are compared for a coarse grid (281 nodes in each direction) and a finer grid (561 nodes in each direction), as shown in *Fig. 7*. MUSCL scheme clearly performs better on the finer grid as compared to the coarser one, however the performance of PINN is unaffected by grid sparsity. In CFD simulations, sometimes it is not possible to have a fine grid due to computational cost, speed, as well as other geometric considerations. The ability of PINN to act as a "meshless solver" is, therefore, a major advantage over traditional numerical schemes.

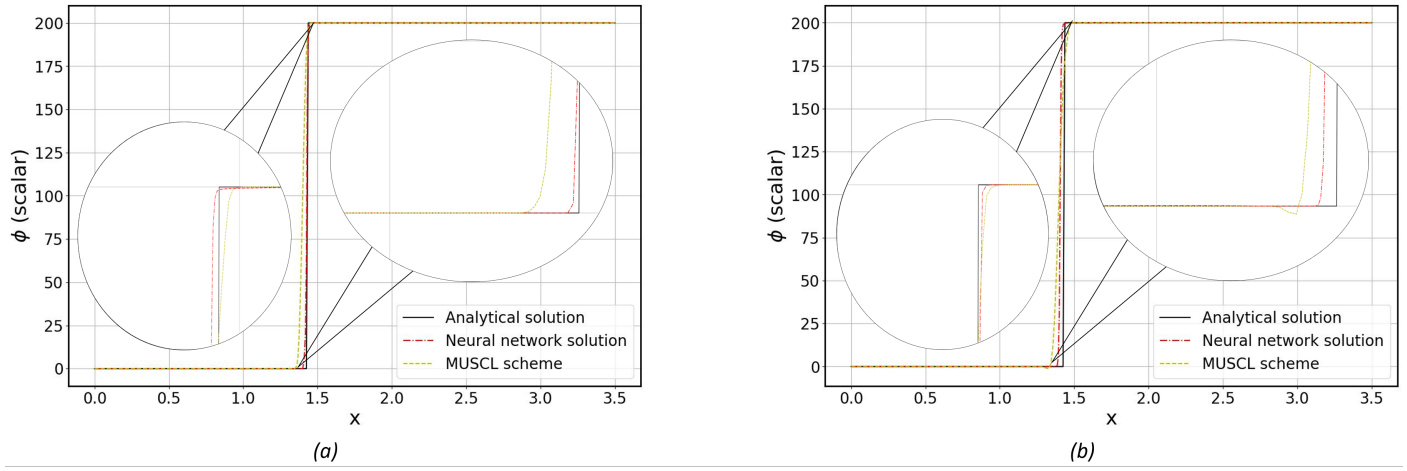

Figure 8: $\phi(x, y)$ vs x used for comparison between solutions to 2D steady state advection-diffusion equation obtained analytically, using PINN, and MUSCL numerical scheme at y = 1.4 on (a) an orthogonal grid, and (b) a non-orthogonal grid with a magnified look at the sharp discontinuities

Often times in CFD simulations, it is not possible to use just hexahedral elements to generate a mesh due to geometric complexities, computational resource limitations, and time limit. In such cases, tetrahedral elements (which are triangular in 2D) are used which leads to the creation of a non-orthogonal grid, in which the entire mesh is not aligned with the flow. In order to better understand the drawbacks of such numerical schemes, a non-orthogonal grid is used to compare the performance of PINN with the MUSCL scheme. The experiment was run and compared for both techniques at y = 1.4 as can be seen in *Fig. 8*. The discontinuities are blown up and displayed in the same figure in order to better compare the performance of PINN and MUSCL. The MUSCL scheme solution suffers from false diffusion due to non-orthogonality of grid,

as well as numerical oscillation due to the step-wise function-like nature of the flow at such low viscosity. In comparison, PINN shows no numerical artifacts and is a much better approximator of the analytical solution.

4.3. 2D Transient advection-diffusion

Since PINN performed well on the first two cases which involved two dimensions (x, t for 1D transient and x,y for 2D steady state), the natural progression is to extrapolate and see how it will perform when the dimensionality of the problem is increased. The way to do this is by evaluating the same model architecture for a 2D transient advection-diffusion case, where the dimensions are now x, y, and t. The model is again trained using all the initial and boundary condition data, as well as the collocation points' PDE residual constraint. Unfortunately, the same model architecture exhibits a lack of convergence with the analytical solution, as shown in *Fig. 9*.

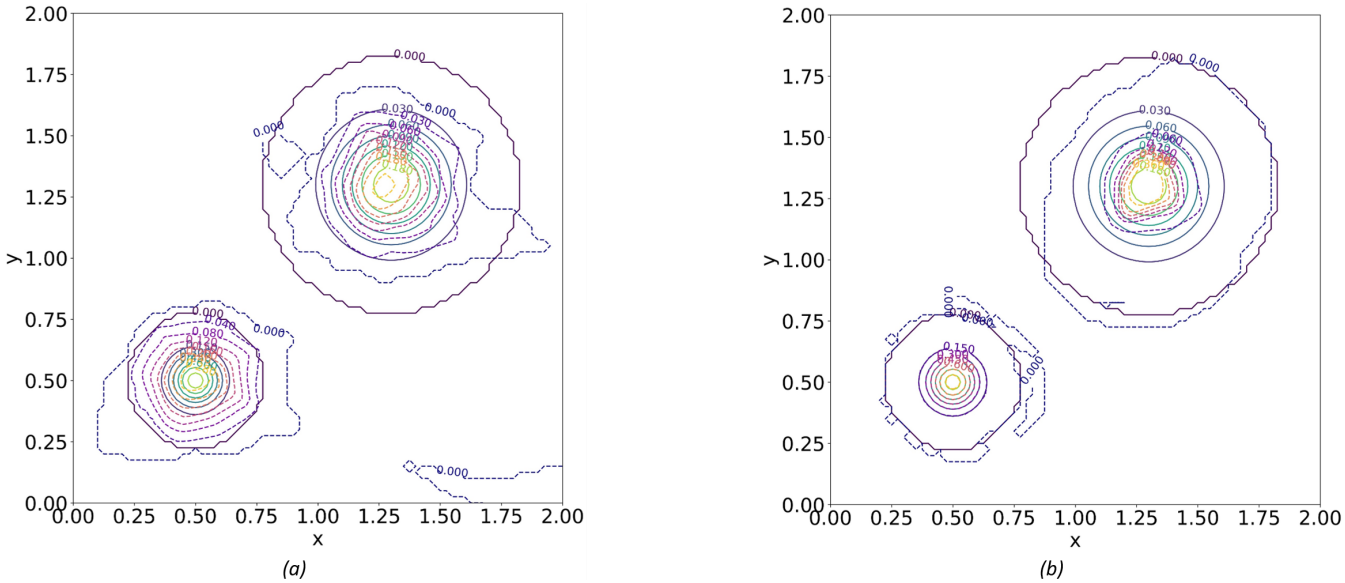

Figure 9: Contour plots of $\phi(t, x, y)$ used for comparison between solutions to 2D steady state advection-diffusion equation obtained analytically and by using PINN at t = 0.0 and t = 1.0 with (a) given DNN architecture, and (b) incorporating I.C. hard-coding

The DNN architecture fails to approximate the initial condition, and the solution gets progressively worse as time increases. In order to understand if the solution can be improved, a technique was employed in which the initial conditions are hard-coded into the problem, and the DNN is left with minimizing a modified loss function which consists only of boundary data loss and PDE loss [34], the results of which can be seen in *Fig. 9*. This modified approach does yield the exact solution at time t = 0, but that is because the nature of the objective function forces it to be so. Once the DNN progresses in time, the solution seems to break down yet again, with the DNN not being able to converge to the analytical solution.

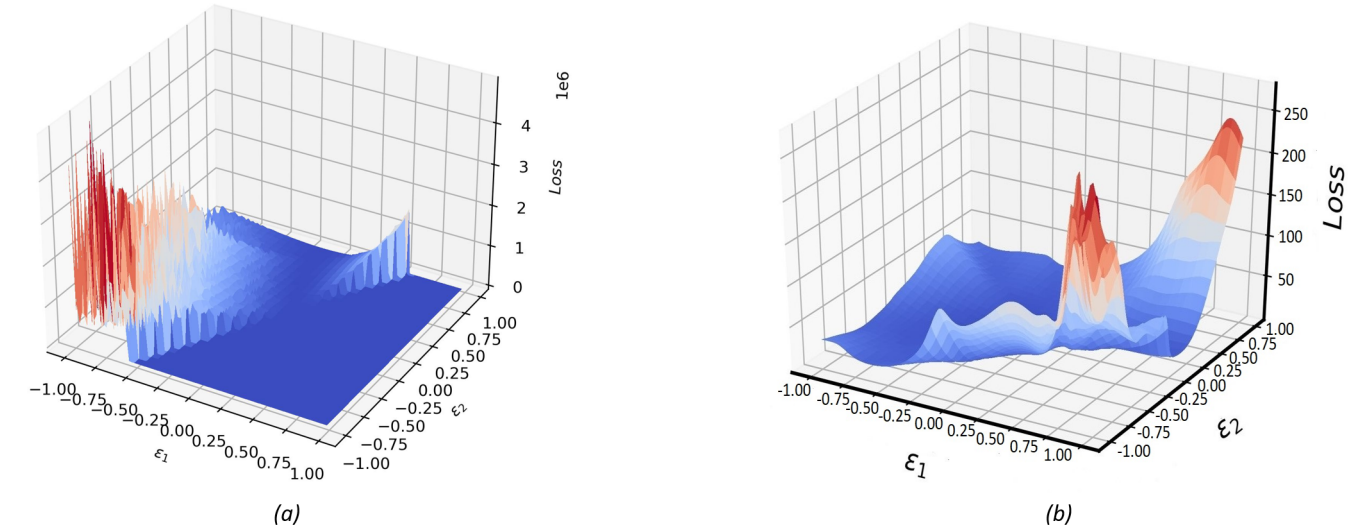

Figure 10: Loss landscape for PINN plotted along first two dominant Hessian eigenvectors ε_1 and ε_2 for (a) Re = 2, and (b) Re = 2 x 10^7

To understand the reasons behind a lack of convergence, a loss landscape is plotted by perturbing the (trained) model across the first two dominant Hessian eigenvectors and computing the corresponding loss values. This is done for 2 different

Reynolds numbers, Re = 2 and Re = 2 x 10^7, as shown in *Fig. 10*. The loss landscape for both Reynolds numbers is complex and non-symmetric. It is very much possible that the optimizer might have gotten stuck in a local minima (minima in a reduced dimension space is most likely a saddle point in the original space) with a very high loss function.

5. Conclusions

A machine learning model with MLPs has been built to predict the value of scalar transport at any point in the fluid domain with minimal supervisory data provided in the form of initial/boundary conditions. Performance of PINN has been compared with that of traditional numerical methods as well as the analytical/closed-form solution for bench-marking purposes. However, the proposed deep learning architecture, which shows promising results for a diverse collection of problems in computational engineering, should not be viewed as replacements of numerical methods that meet the standards for robustness and computational efficiency. Rather, PINNs provide intuition for algorithmic development of meshless solvers, which can serve as a viable alternative.

- For lower dimension problems, deep learning solver outperforms many traditional numerical solvers with ease.
- For 1D transient advection equation, CFL condition dictates performance of numerical schemes. Due to dependency on the extent of spatial discretization, these numerical methods cannot be used freely. first-order upwind schemes exhibit numerical diffusion and tend to greatly smear the moving steep fronts. This artificially introduced diffusion is even more problematic in this case, since this is purely an advection problem. Unlike first order upwind, second-order central difference scheme largely overcomes the problem of numerical diffusion but exhibits numerical oscillations when it starts skipping cells. On the other hand, PINN is invariant to spatial discretization and displays unconditional stability as well as maximum accuracy.
- For 2D steady state advection-diffusion equation, grid sparsity and alignment dictates performance of numerical schemes. As mesh grid size is refined, the residual error in the numerical schemes goes down. However, it might not be possible in some CFD simulations to refine mesh size, in which case this becomes a liability. Relying solely on grid refinement for dealing with numerical oscillations is often impractical because of the related computational cost. Also, if the flow is not aligned with the grid, i.e., if the grid is non-orthogonal, numerical oscillations are introduced at sharp discontinuities and numerical diffusion already present in the solution gets accentuated. Again, there are cases where we are forced to use tetrahedral elements (which are triangular elements in 2D) instead of hexahedral (quad) elements, which contribute to increased numerical artifacts in these solutions. A lack of symmetric element faces in triangular mesh also contributes to errors in numerical schemes. PINN performs much better in comparison, when it comes to grid size variation, and shows very little drop in accuracy for non-orthogonal grids.
- For 2D transient advection-diffusion equation, PINN architecture that was used in the first two cases does not exhibit the same level of accuracy. After inspecting the loss landscape for this case, it can be hypothesized that the optimizer might be getting stuck in local saddle points which might cause errors in the PINN solution. Another possibility is the fact that a majority of the nodes in the domain are at functional value, ϕ=0 due to Dirichlet boundary conditions, which could prompt the optimizer to consider a trivial solution as the one with the least error.
- Deep Learning framework capable of solving 1D transient and 2D steady state problems exhibits difficulty in solving 2D transient problems (increase in dimensionality) and requires significant user intervention to obtain a plausible solution.
- If this architecture can be made generalizable, it would be a timely contribution to specific applications such as data-driven forecasting of physical processes, model predictive control, and multi-physics modeling.

6. Acknowledgements

Funding provided by Carnegie Bosch Institute grant. Discussions with Prof. Aarti Singh and Zhiyuan Zhao from the Carnegie Mellon Machine Learning department provided insights into deep learning methodologies and utilizing PINN for Ordinary Differential Equations.

References

[1] 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.

[2] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Communications of the ACM 60 (6) (2017) 84–90.

[3] 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 (2017).

[4] M. Raissi, G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, Journal of Computational Physics 357 (2018) 125–141.

[5] 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.

[6] J. Han, A. Jentzen, et al., Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Communications in mathematics and statistics 5 (4) (2017) 349–380.

[7] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, M. W. Mahoney, Characterizing possible failure modes in physics-informed neural networks, Advances in Neural Information Processing Systems 34 (2021) 26548–26560.

[8] Z. Yao, A. Gholami, Q. Lei, K. Keutzer, M. W. Mahoney, Hessian-based analysis of large batch training and robustness to adversaries, Advances in Neural Information Processing Systems 31 (2018).

[9] E. Lotfi, M.-R. Akbarzadeh-T, A novel single neuron perceptron with universal approximation and xor computation properties, Computational intelligence and neuroscience 2014 (2014).

[10] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366.

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

[12] S. Singh, D. You, A multi-block adi finite-volume method for incompressible navier–stokes equations in complex geometries, Journal of Computational Physics 230 (19) (2011) 7400–7417.

[13] M. Wu, S. Singh, Predictions of flow field around bluff bodies with machine learning (ml) models trained using cfd simulations (2020).

[14] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al., Mlp-mixer: An all-mlp architecture for vision, Advances in Neural Information Processing Systems 34 (2021) 24261–24272.

[15] J. Sola, J. Sevilla, Importance of input data normalization for the application of neural networks to complex industrial problems, IEEE Transactions on nuclear science 44 (3) (1997) 1464–1468.

[16] 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.

[17] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International conference on machine learning, PMLR, 2015, pp. 448–456.

[18] B. Hanin, Which neural net architectures give rise to exploding and vanishing gradients?, Advances in neural information processing systems 31 (2018).

[19] S.-i. Amari, Backpropagation and stochastic gradient descent method, Neurocomputing 5 (4-5) (1993) 185–196.

[20] S. Khirirat, H. R. Feyzmahdavian, M. Johansson, Mini-batch gradient descent: Faster convergence under data sparsity, in: 2017 IEEE 56th Annual Conference on Decision and Control (CDC), IEEE, 2017, pp. 2880–2887.

[21] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[22] R. Courant, K. Friedrichs, H. Lewy, Über die partiellen differenzengleichungen der mathematischen physik, Mathematische annalen 100 (1) (1928) 32–74.

[23] H. Abdi, L. J. Williams, Principal component analysis, Wiley interdisciplinary reviews: computational statistics 2 (4) (2010) 433–459.

[24] Z. Yao, A. Gholami, K. Keutzer, M. W. Mahoney, Pyhessian: Neural networks through the lens of the hessian, in: 2020 IEEE international conference on big data (Big data), IEEE, 2020, pp. 581–590.

[25] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th annual international conference on machine learning, 2009, pp. 41–48.

[26] J. Crank, P. Nicolson, A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type, in: Mathematical proceedings of the Cambridge philosophical society, Vol. 43, Cambridge University Press, 1947, pp. 50–67.

[27] P. Stampolidis, M. C. Gousidou-Koutita, A computational study with finite difference methods for second order quasilinear hyperbolic partial differential equations in two independent variables, Applied Mathematics 9 (11) (2018) 1193–1224.

[28] B. P. Leonard, A stable and accurate convective modelling procedure based on quadratic upstream interpolation, Computer methods in applied mechanics and engineering 19 (1) (1979) 59–98.

[29] G. D. Van Albada, B. v. Leer, W. Roberts, A comparative study of computational methods in cosmic gas dynamics, in: Upwind and high-resolution schemes, Springer, 1997, pp. 95–103.

[30] B. Van Leer, Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method, Journal of computational Physics 32 (1) (1979) 101–136.

[31] M. Ciardi, P. Sagaut, M. Klein, W. Dawes, A dynamic finite volume scheme for large-eddy simulation on unstructured grids, Journal of Computational Physics 210 (2) (2005) 632–655.

[32] N. P. Waterson, H. Deconinck, Design principles for bounded higher-order convection schemes–a unified approach, Journal of Computational Physics 224 (1) (2007) 182–207.

[33] B. Van Leer, P. Woodward, The muscl code for compressible flow: philosophy and results, TICOM Conferece (1979).

[34] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE transactions on neural networks 9 (5) (1998) 987–1000.