南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

Neural-PDE: A RNN based neural network for solving time dependent PDEs

YIHAO HU, TONG ZHAO, SHIXIN XU, LIZHEN LIN, AND ZHILIANG XU

ICML 2022

Background

南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

The research of time-dependent partial differential equations (PDEs) is regarded as one of the most important disciplines in applied mathematics. PDEs appear ubiquitously in a broad spectrum of fields including physics, biology, chemistry, and finance, to name a few.

A time-dependent partial differential equation is an equation of the form:

$$u_t = f(x_1, \cdots, u, \frac{\partial u}{\partial x_1}, \cdots, \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial x_1 \partial x_1}, \cdots, \frac{\partial^2 u}{\partial x_1 \partial x_n}, \cdots, \frac{\partial^n u}{\partial x_1 \cdots \partial x_n})$$

where $\mathrm{u} = \mathrm{u}(x_1, \ldots, x_n, \mathrm{t})$ is unknown, $x_i \in R$ are spatial variables, and the operator f maps $R^N \mapsto R$.

Background

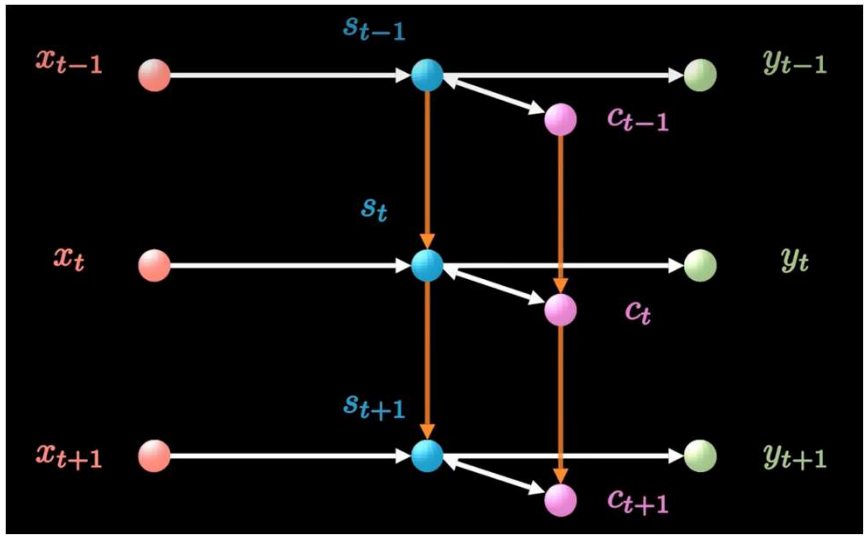
The conventional approaches, such as **finite element method** (Bathe, 2007) or **pseudospectral method** (Fornberg, 1998), suffer from high computational costs in constructing meshes for high-dimensional PDEs. With the development of scientific machine learning, Physics-informed neural networks (PINNs) (Lagaris et al., 1998; Raissi et al., 2019) have emerged as a promising novel approach. Conventional PINNs and most variants employ multilayer perceptrons (MLP) as end-toend frameworks for point-wise predictions, achieving remarkable success in various scenarios.
But conventional PINNs, largely relying on MLP-based architecture, can **overlook important temporal dependencies in real-world physical systems**.

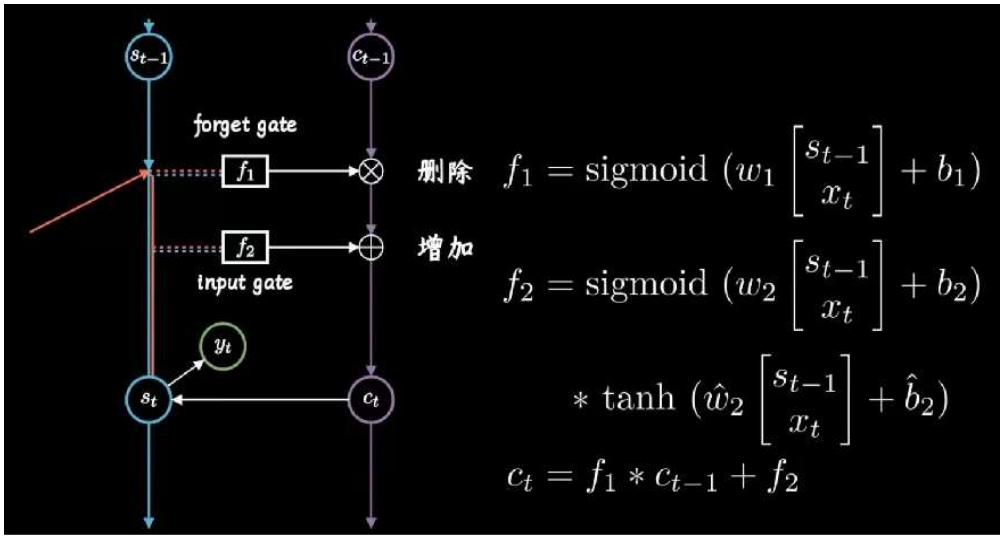
Long Short-Term Memory (LSTM) is a neural network built upon RNNs. Unlike vanilla RNNs, which suffer from losing long term information and high probability of gradient vanishing or exploding, LSTM has a specifically designed memory cell with a set of new gates such as input gate and forget gate. Equipped with these new gates which control the time to preserve and pass the information, LSTM is capable of learning **long term dependencies** without the danger of having gradient vanishing or exploding.

Background

Inspired by numerical PDE schemes and LSTM neural network, we propose a new deep learning framework, denoted as **Neural-PDE**. It simulates multi-dimensional governing laws, represented by time-dependent PDEs, from time series data generated on some grids and predicts the next n time steps data.

Long Short-Term Memory networks (LSTM):





Background

Long Short-Term Memory networks (LSTM):

$$
\begin{aligned}
\boldsymbol{i}_t &= \sigma(\mathbf{W}_i^{(x)}\boldsymbol{x}_t + \mathbf{W}_i^{(h)}\boldsymbol{h}_{t-1} + \mathbf{W}_i^{(c)}\boldsymbol{c}_{t-1} + \boldsymbol{b}_i) \ , \\
\boldsymbol{f}_t &= \sigma(\mathbf{W}_f^{(x)}\boldsymbol{x}_t + \mathbf{W}_f^{(h)}\boldsymbol{h}_{t-1} + \mathbf{W}_f^{(c)}\boldsymbol{c}_{t-1} + \boldsymbol{b}_f) \ , \\
\boldsymbol{c}_t &= \boldsymbol{f}_t\boldsymbol{c}_{t-1} + \boldsymbol{i}_t \tanh(\mathbf{W}_c^{(x)}\boldsymbol{x}_t + \mathbf{W}_c^{(h)}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c) \ , \\
\boldsymbol{o}_t &= \sigma(\mathbf{W}_o^{(x)}\boldsymbol{x}_t + \mathbf{W}_o^{(h)}\boldsymbol{h}_{t-1} + \mathbf{W}_o^{(c)}\boldsymbol{c}_t + \boldsymbol{b}_o), \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \tanh(\boldsymbol{c}_t) \ ,
\end{aligned}
$$

where σ is the logistic sigmoid function, $\boldsymbol{W}$s are weight matrices, $\boldsymbol{b}$s are bias vectors, and subscripts $\boldsymbol{i}$, $\boldsymbol{f}$, $\boldsymbol{o}$ and $\boldsymbol{c}$ denote the input gate, forget gate,output gate and cell vectors respectively, all of which have the same size as hidden vector $\boldsymbol{h}$.

Mathematical Motivation

Recurrent neural network including **LSTM** is an artificial neural network structure of the form:

$$\boldsymbol{h}^t = \sigma(\mathbf{W}^{hx}\boldsymbol{x}^t + \mathbf{W}^{hh}\boldsymbol{h}^{t-1} + \boldsymbol{b}_h) \equiv \sigma_a(\boldsymbol{x}^t, \boldsymbol{h}^{t-1}) \equiv \sigma_b(\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^t)$$

$$\begin{aligned} \boldsymbol{y}^t &= \sigma(\mathbf{W}^{hy}\boldsymbol{h}^t + \boldsymbol{b}_y) \\ &\equiv \sigma_c(\boldsymbol{h}^t) \equiv \sigma_d(\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^t) \ . \end{aligned}$$

Canonical structure for such recurrent neural network usually calculates the current state value by its previous time step value $\boldsymbol{h}^{t-1}$ and current state input $\boldsymbol{x}^t$. Similarly, in numerical PDEs, the next step data at a grid point is updated from the previous (and current) values on its nearby grid points.

Mathematical Motivation

Thus, what if we replace the temporal input $\boldsymbol{h^{t-1}}$ and $\boldsymbol{x^t}$ with spatial information? A simple sketch of the upwinding method for a $\boldsymbol{1D}$ example of $\boldsymbol{u(x;\ t)}$:

$$u_t + \nu u_x = 0$$

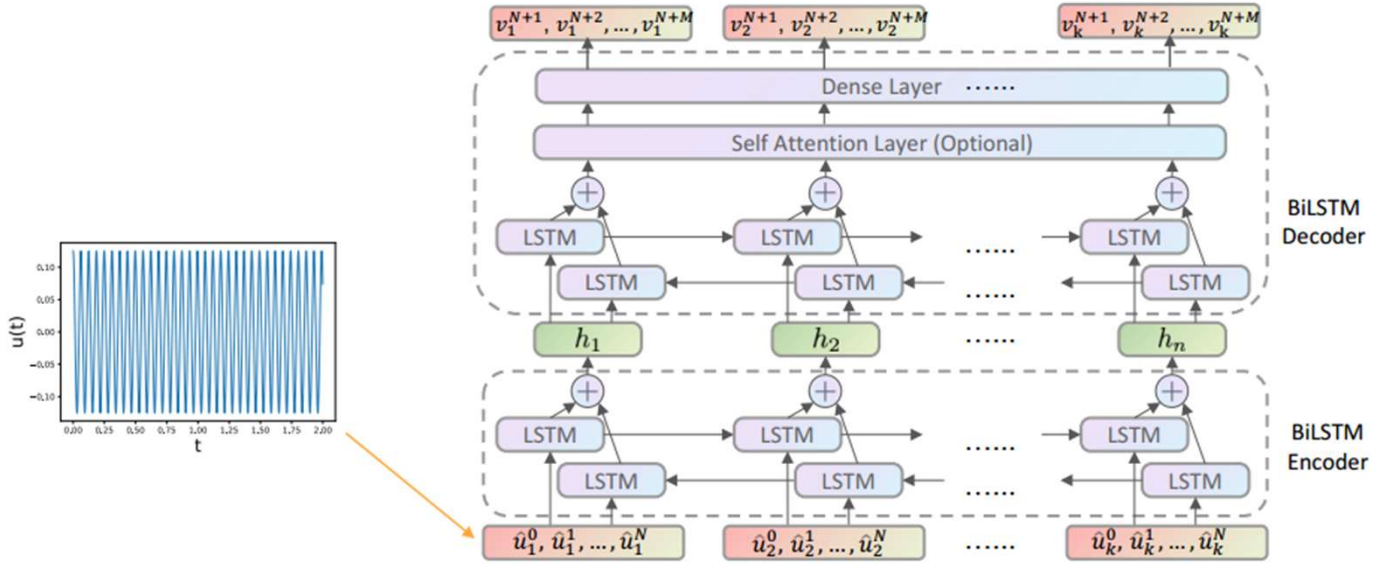
$$\begin{aligned} u_i^{n+1} &= u_i^n - \nu\frac{\delta t}{\delta x}(u_i^n - u_{i-1}^n) + \mathcal{O}(\delta x, \delta t) \rightarrow \hat{u}_i^{n+1} = f_2(\hat{u}_{i-1}^n, \hat{u}_i^n) \\ &\equiv f_\theta\big(f_\eta(\boldsymbol{x}_i, \boldsymbol{h}_{i-1}(u))\big) = f_{\theta,\eta}\big(\hat{u}_0^n, \hat{u}_1^n, \cdots, \hat{u}_{i-1}^n, \hat{u}_i^n\big) = v_i^{n+1} \\ \boldsymbol{x}_i &= \hat{u}_i^n,\ \boldsymbol{h}_{i-1}(\hat{u}) = \sigma(\hat{u}_{i-1}^n, \boldsymbol{h}_{i-2}(\hat{u})) \equiv f_\eta(\hat{u}_0^n, \hat{u}_1^n, \hat{u}_2^n, \cdots, \hat{u}_{i-1}^n). \end{aligned}$$

$$\hat{u}_i^n \approx u(i\delta x, n\delta t) \quad f_2 \equiv \hat{u}_i^n - \nu\tfrac{\delta t}{\delta x}(\hat{u}_i^n - \hat{u}_{i-1}^n) : \mathbb{R}^2 \rightarrow \mathbb{R}$$

$\boldsymbol{f_\theta}$ represents the dynamics of the hidden layers in decoder with parameters $\boldsymbol{\theta}$, and $\boldsymbol{f_\eta}$ specifies the dynamics of the LSTM layer in encoder withe parameters $\boldsymbol{\eta}$. The function $\boldsymbol{f_{\theta,\eta}}$ simulates the dynamics of the Neural-PDE with paramaters $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$.

Method

Neural-PDE



$$\boldsymbol{h}_i = \overrightarrow{\mathrm{LSTM}}(\boldsymbol{a}_i) \oplus \overleftarrow{\mathrm{LSTM}}(\boldsymbol{a}_i),$$

$$v_i = \left(\overrightarrow{\mathrm{LSTM}}(\boldsymbol{h}_i) \oplus \overleftarrow{\mathrm{LSTM}}(\boldsymbol{h}_i)\right) \cdot \mathbf{W},$$

$$\mathcal{L} = \sum_{t=N+1}^{N+M} \sum_{i=1}^{k} ||\hat{u}_i^t - v_i^t||^2 \ ,$$

In particular, we use the bidirectional LSTM to better retain the state information from data on grid points which are neighbourhoods in the mesh but far away in input matrix.

Method

For a $\boldsymbol{n}$-dimensional time-dependent partial differential equation with $\boldsymbol{K}$ collocation points, the input and output data for $\boldsymbol{t\epsilon(0,T)}$ will be of the form:

$$\boldsymbol{A}(K,N) = \begin{bmatrix} \boldsymbol{a}_0^N \\ \vdots \\ \boldsymbol{a}_\ell^N \\ \vdots \\ \boldsymbol{a}_K^N \end{bmatrix} = \begin{bmatrix} \hat{u}_0^0 & \hat{u}_0^1 & \cdots & \hat{u}_0^n & \cdots & \hat{u}_0^N \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hat{u}_\ell^0 & \hat{u}_\ell^1 & \cdots & \hat{u}_\ell^n & \cdots & \hat{u}_\ell^N \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hat{u}_K^0 & \hat{u}_K^1 & \cdots & \hat{u}_K^n & \cdots & \hat{u}_K^N \end{bmatrix}$$

$$\boldsymbol{B}(K,M) = \begin{bmatrix} \boldsymbol{b}_0^M \\ \vdots \\ \boldsymbol{b}_\ell^M \\ \vdots \\ \boldsymbol{b}_K^M \end{bmatrix} = \begin{bmatrix} v_0^{N+1} & v_0^{N+2} & \cdots & v_0^{N+m} & \cdots & v_0^{N+M} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ v_\ell^{N+1} & v_\ell^{N+2} & \cdots & v_\ell^{N+m} & \cdots & v_k^{N+M} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ v_K^{N+1} & v_K^{N+2} & \cdots & v_K^{N+m} & \cdots & v_K^{N+M} \end{bmatrix}$$

By adding Bidirectional **LSTM** encoder in the **Neural-PDE**, it will automatically extract the information from the time series data as:

$$\boldsymbol{B}(K,M) = PDESolver(\boldsymbol{A}(K,N)) = PDESolver(\boldsymbol{a}_0^N, \boldsymbol{a}_1^N, \cdots \boldsymbol{a}_i^N, \cdots, \boldsymbol{a}_K^N)$$

Experiment

南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

Table 1: Error analysis models

	Wave	Heat	Burgers'
Equation	$u_{tt} = \frac{1}{16\pi^2} u_{xx}$	$u_t = u_{xx}$	$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0.1\frac{\partial^2 u}{\partial x^2}$
IC	$\sin(4\pi x)$	$6\sin(\pi x)$	$u(0 \le x \le L, t = 0) = 0.9$
BC	*periodic*	*periodic*	*periodic*

Table 2: L^2 error for model evaluation

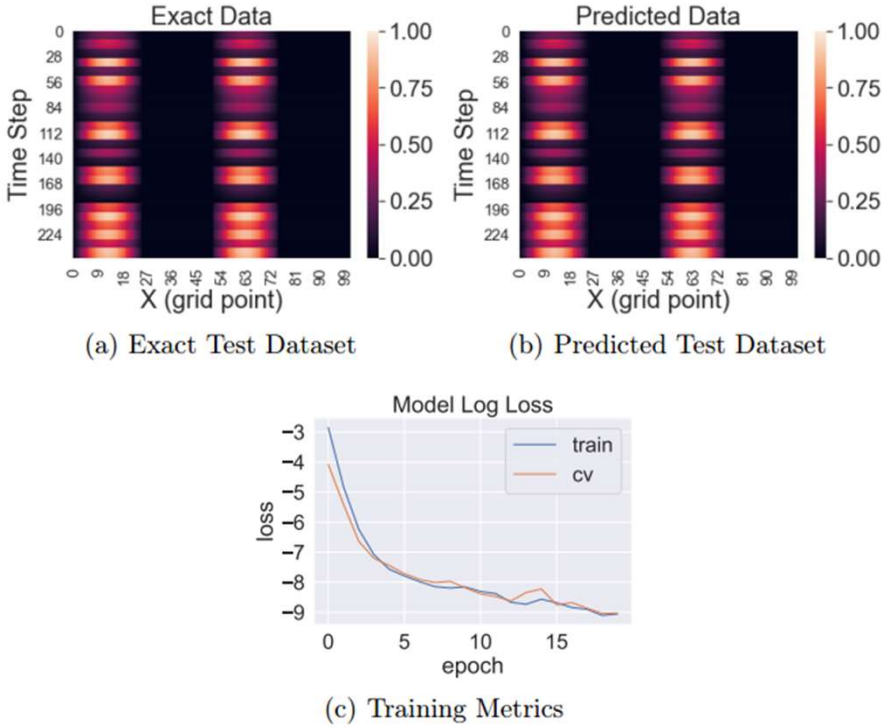
$\Delta x = 0.1$	Wave	Heat	Burgers'
$\Delta t = 0.1$	4.385×10^{-3}	6.912×10^{-5}	9.450×10^{-4}
$\Delta t = 0.01$	3.351×10^{-5}	5.809×10^{-5}	5.374×10^{-3}
$\Delta t = 0.001$	1.311×10^{-5}	3.757×10^{-5}	1.244×10^{-3}

$$\boldsymbol{B}(K, 10) = PDESolver(\boldsymbol{A}(K, 30))$$

We used the Neural-PDE which only consists of 3 layers: 2 bi-lstm (encoder-decoder) layers with 20 neurons each and 1 dense output layer with 10 neurons and achieved MSEs from $O(10^{-3})$ to $O(10^{-5})$ within 20 epochs, a MLP based neural network such as Physical Informed Neural Network usually will have more layers and neurons to achieve similar L^2 errors.

Experiment

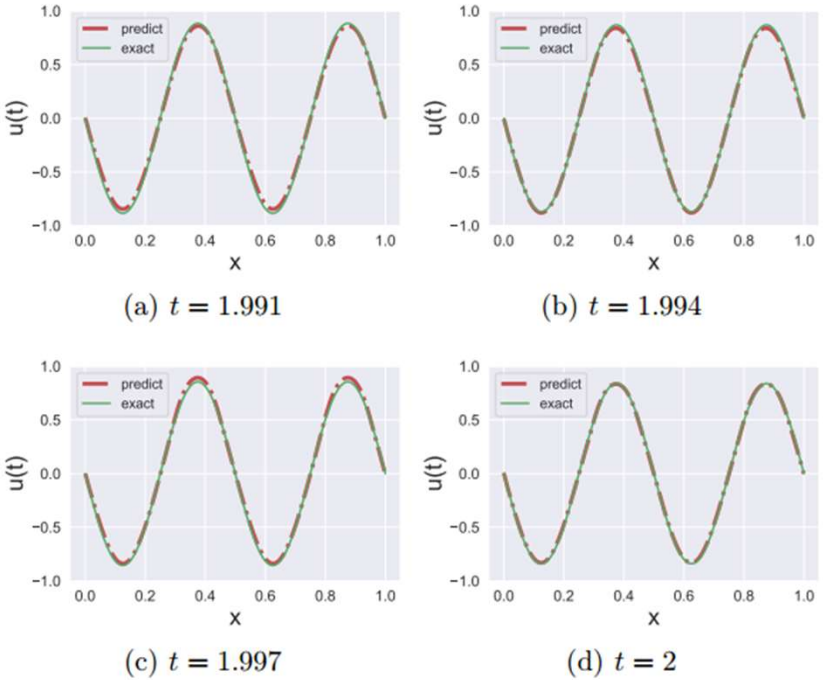
Wave equation



(a) Exact Test Dataset (b) Predicted Test Dataset

(c) Training Metrics

Figure 4: The Neural-PDE for solving the wave equation.



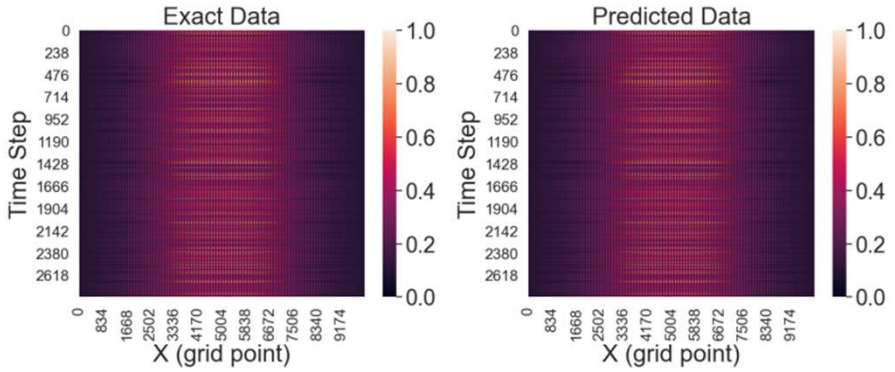
(a) $t = 1.991$ (b) $t = 1.994$

(c) $t = 1.997$ (d) $t = 2$

Figure 5: Comparison between exact solution and Neural-PDE prediction of the 1D wave equation at various time points.

Experiment

Heat equation



(a) Exact Test Dataset (b) Predicted Test Dataset

Figure 6: Heatmaps of the heat equation data. $\delta x = 0.02, \delta y = 0.02$, $\delta t = 10^{-4}$, MSE: 2.1551×10^{-6}.

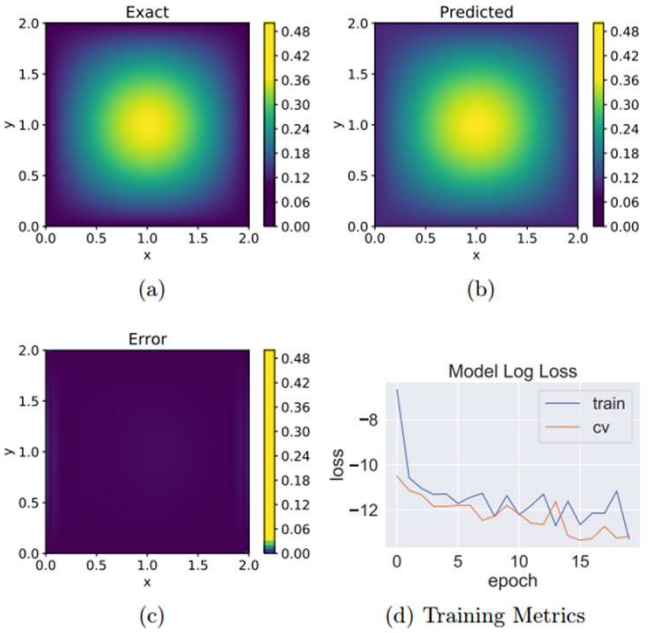
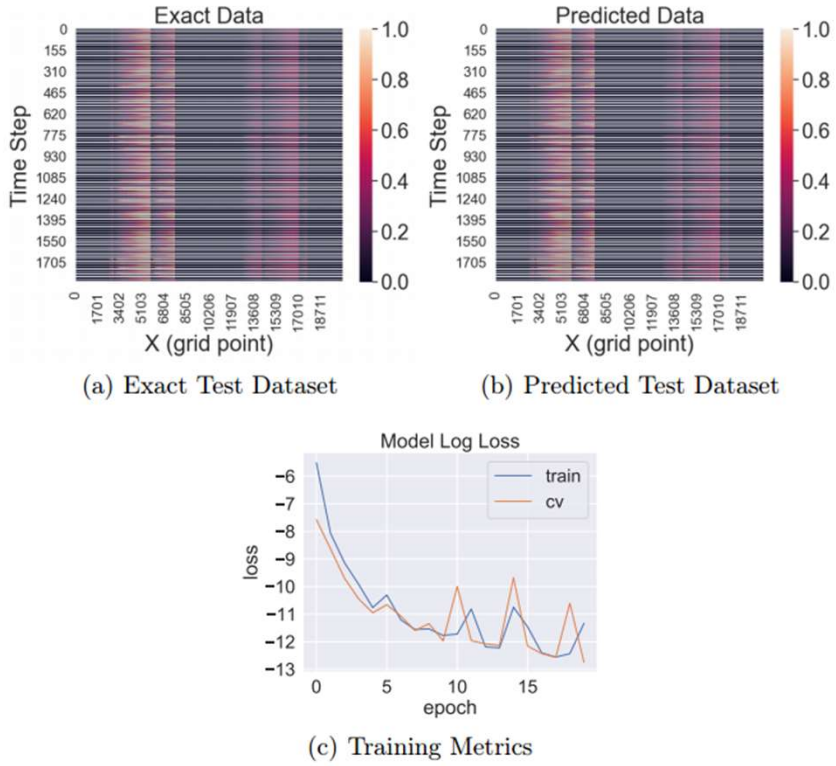


Figure 7: The Neural-PDE for solving the 2D heat equation. (a) is the exact solution $u(x, y, t = 0.15)$ at the final state. (b) is the Neural-PDE prediction. (c) is the corresponding error map and (d) shows the training and cross-validation errors.

Experiment

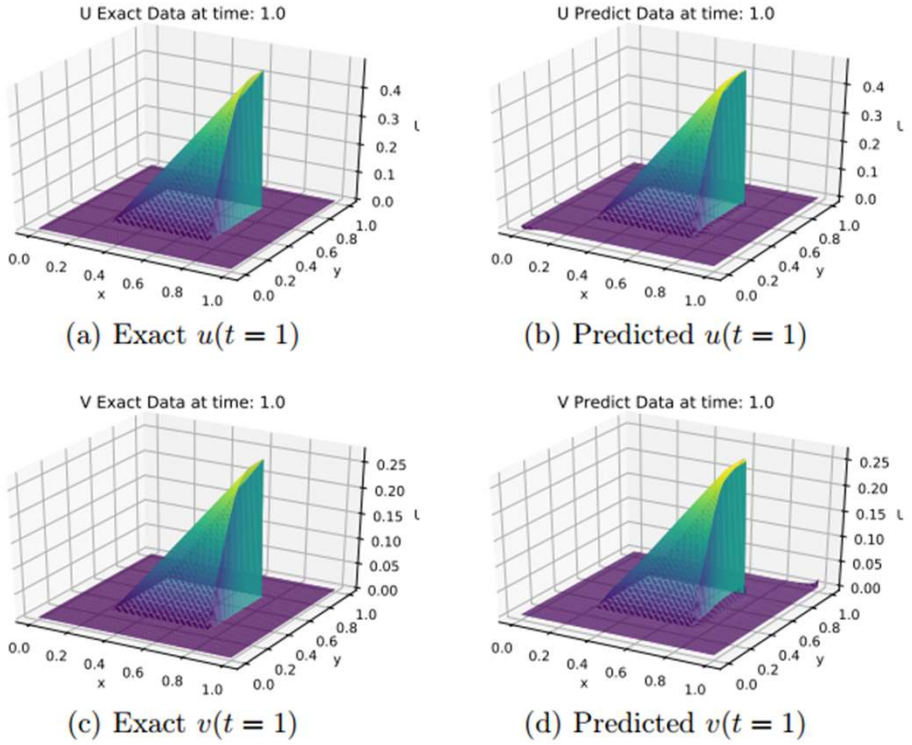
Inviscid Burgers' Equation



(a) Exact Test Dataset (b) Predicted Test Dataset

(c) Training Metrics

Figure 8: Neural-PDE prediction on the $2D$ Burgers' equation.



(a) Exact $u(t = 1)$ (b) Predicted $u(t = 1)$

(c) Exact $v(t = 1)$ (d) Predicted $v(t = 1)$

Figure 9: Neural-PDE shows accurate prediction on Burgers' equation.

Thanks