Implicit Form Neural Network for Learning Scalar Hyperbolic Conservation Laws

Xiaoping Zhang [1], Tao Cheng [1] and Lili Ju [2]

[1]School of Mathematics and Statistics, Wuhan University
[2]Department of Mathematics, University of South Carolina

Mathematical and Scientific Machine Learning, 2021
August 19, 2021

In recent years, deep learning techniques have been applied to mathematical and scientific computing problems, and a large number of excellent research works on learning solutions of PDEs have emerged.

- The first attempt of solving PDEs using NNs can be traced back at least to the late last century. [1]. However, due to the limitation of computational resource back that time, it unfortunately did not attract much attention of researchers.
- Currently, there are two major ways to use deep learning for numerical solutions of PDEs:
  1. **Deep Ritz method [2] and its variants** Convert the PDEs into their equivalent variational forms and then solve them using NNs
  2. **PINN [3]and its variants** Use NNs to directly deal with the original PDEs

[1] Lee and Kang, J. Comput. Phys. 1990 .
[2] E and Yu, Comm. Math. Stat. 2018
[3] Raissi et al., J. Comput. Phys. 2019

Scalar hyperbolic conservation law

Let $\boldsymbol{x} = (x_1, \cdots, x_d)^T \in \mathbb{R}^d$ and $\boldsymbol{f} = (f_1, \cdots, f_d)^T \in \mathbb{R}^d$, the scalar hyperbolic conservation law can be formulated as follows :

$$\begin{cases} u_t + \nabla \cdot \boldsymbol{f}(u) = 0, \quad (\boldsymbol{x}, t) \in \mathbb{R}^d \times (0, T], & \text{(1a)} \\ u(\boldsymbol{x}, 0) = \phi(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d. & \text{(1b)} \end{cases}$$

where u is an unknown function defined in $\mathbb{R}^d$.

Remark

Although the deep Ritz method and PINN have achieved great success, they still exhibit poor performance in handling problems with strong discontinuous solutions, such as hyperbolic conservation laws.

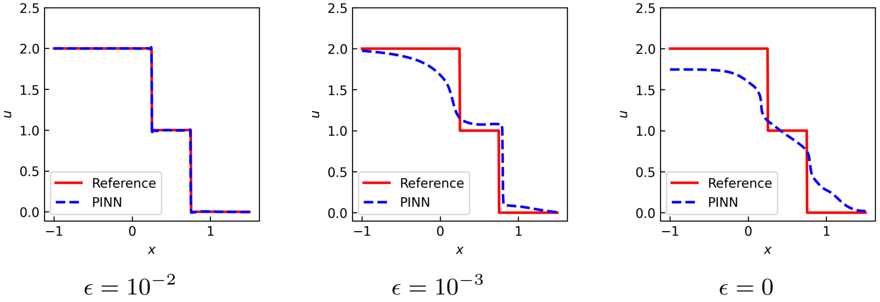


Figure: Reference solution and predicted solution of PINN for Burgers' equation $u_t + uu_x - \epsilon u_{xx} = 0$ when $t = 0.5$

IFNN

Theorem

An implicit form for the solution of (1) *can be formulated as*

$$u = \phi(\boldsymbol{x} - \boldsymbol{f}'(u)t), \tag{2}$$

where $\boldsymbol{f}'$ *denotes the velocity*

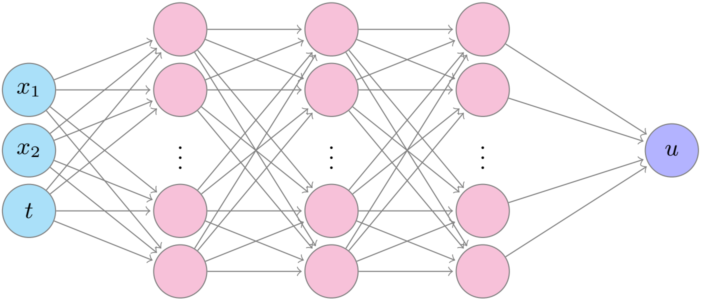
$$\boldsymbol{f}'(u) = (f_1'(u), \cdots, f_d'(u))^T. \tag{3}$$

Contribution

A fully-connected neural network 'IFNN' is proposed to learn the solution of (1) by using the loss function based on the implicit form (2) instead of the original equation (1).

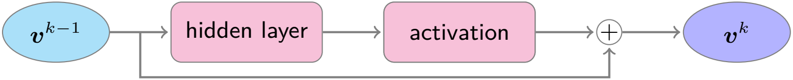
- Network model with parameters $\boldsymbol{\Theta} = \{\boldsymbol{W}^k, \boldsymbol{b}^k\}_{k=1}^{D}$.

$$u(\boldsymbol{x}, t; \Theta) = (\ell_D \circ \sigma \circ \ell_{D-1} \cdots \circ \sigma \circ \ell_1)(\boldsymbol{x}, t), \quad \ell_k(\boldsymbol{v}^{k-1}) = \boldsymbol{W}^k \boldsymbol{v}^{k-1} + \boldsymbol{b}^k. \quad (4)$$



- Residual block

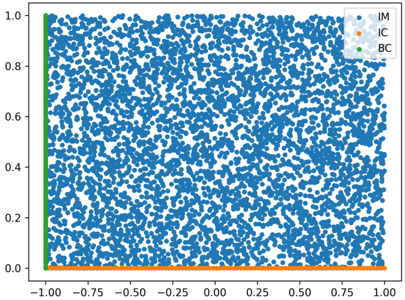
$$\boldsymbol{v}^k = \boldsymbol{v}^{k-1} + \sigma \circ \ell_k(\boldsymbol{v}^{k-1})$$



Sampling stragegy

To train the network (4), we need a set of sampling points with respect to $(\boldsymbol{x}, t)$, and the Latin Hypercubic Sampling (LHS) is used to generate the point set $\mathcal{D}$, and then split it into three parts:

- $\mathcal{D}_{\mathrm{IC}} = \mathcal{D} \cap (\Omega \times \{0\})$,
- $\mathcal{D}_{\mathrm{BC}} = \mathcal{D} \cap (\partial\Omega \times (0, T])$,
- $\mathcal{D}_{\mathrm{IM}} = \mathcal{D} \backslash (\mathcal{D}_{\mathrm{IC}} \cup \mathcal{D}_{BC})$.



Loss function

$$\mathcal{L}(\Theta) = \mathcal{L}_{\mathrm{IM}}(\Theta) + \lambda_1 \mathcal{L}_{\mathrm{IC}}(\Theta) + \lambda_2 \mathcal{L}_{\mathrm{BC}}(\Theta), \tag{5}$$

where

$$\mathcal{L}_{\mathrm{IM}}(\Theta) = \frac{1}{|\mathcal{D}_{\mathrm{IM}}|} \sum_{(\boldsymbol{x},t)\in\mathcal{D}_{\mathrm{IM}}} \left[u(\boldsymbol{x},t;\Theta) - \phi(\boldsymbol{x} - \boldsymbol{f}'(u)t)\right]^2, \tag{6}$$

$$\mathcal{L}_{\mathrm{IC}}(\Theta) = \frac{1}{|\mathcal{D}_{\mathrm{IC}}|} \sum_{(\boldsymbol{x},t)\in\mathcal{D}_{\mathrm{IC}}} \left[u(\boldsymbol{x},t;\Theta) - \phi(\boldsymbol{x})\right]^2. \tag{7}$$

$$\mathcal{L}_{\mathrm{BC}}(\Theta) = \frac{1}{|\mathcal{D}_{\mathrm{BC}}|} \begin{cases} \displaystyle\sum_{(\boldsymbol{x},t)\in\mathcal{D}_{\mathrm{BC}}} \left[u(\boldsymbol{x},t;\Theta) - g_d(\boldsymbol{x},t)\right]^2, & \text{Dirchlet B.C.} \\ \displaystyle\sum_{(\boldsymbol{x},t)\in\mathcal{D}_{\mathrm{BC}}} \left[\frac{\partial u}{\partial n}(\boldsymbol{x},t;\Theta) - g_n(\boldsymbol{x},t)\right]^2, & \text{Neumann B.C.} \\ \displaystyle\sum_{(\boldsymbol{x},t)\in\mathcal{D}_{\mathrm{BC}}} \left[u(\boldsymbol{x},t;\Theta) - u(\boldsymbol{x}',t;\Theta)\right]^2, & \text{Periodic B.C.} \end{cases} \tag{8}$$

where $\boldsymbol{x}'$ is the symmetric point of $\boldsymbol{x}$ in the opposite boundary side of Ω.

Implementation Detail

- Our IFNN is implemented using PyTorch and we use 6 hidden layers with 20 neurons per each hidden layer for all tests.
- Adam optimizer is used to train the network and the number of epochs is set as 25000. The learning rate is initially set to 0.001, and then adjusted with a decay rate of 0.7 per 3000 epochs.
- The hyperparamters λ_1 and λ_2 in the loss function are both chosen as 1.

Numerical Experiments - 1D inviscid Burgers' equation with shock wave (I)

IC: $\phi(x) = \begin{cases} 1, & x \leqslant 0 \\ 0, & x > 0 \end{cases}$, Dirichlet BC: $u(-1, t) = 1$

Model \ Activation	tanh	sin	ReLU
PINN	1.17×10^{-1}	1.54×10^{-1}	1.05×10^{0}
IFNN	2.53×10^{-4}	6.63×10^{-2}	8.01×10^{-2}

Table: L_2 errors of IFNN with different activation functions

$\#l$ \ $\#n$	10	20	30
2	5.76×10^{-2}	2.33×10^{-2}	9.23×10^{-3}
4	8.91×10^{-3}	2.72×10^{-3}	6.71×10^{-4}
6	2.97×10^{-3}	2.50×10^{-4}	2.32×10^{-4}

Table: L_2 errors of IFNN with different number of hidden layers ($\#l$) and neurons ($\#n$).

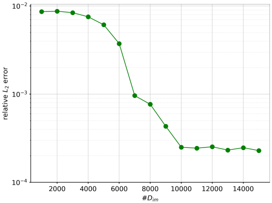
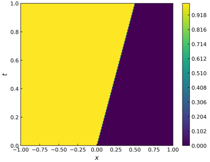


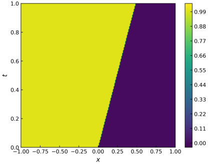
Figure: The effect of the size of training set on the prediction accuracy of IFNN.

Numerical Experiments - 1D inviscid Burgers' equation with shock wave (I)

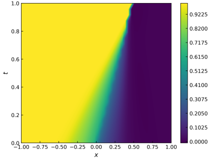
IC: $\phi(x) = \begin{cases} 1, & x \leqslant 0 \\ 0, & x > 0 \end{cases}$, Dirichlet BC: $u(-1,t) = 1$



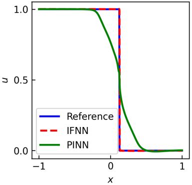
(a) reference solution



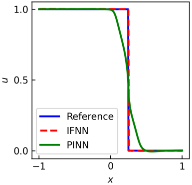
(b) predict solution by IFNN



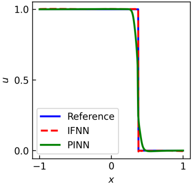
(c) predict solution by PINN



(d) $t = 0.25$



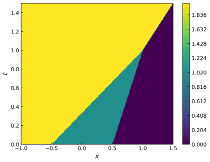
(e) $t = 0.5$



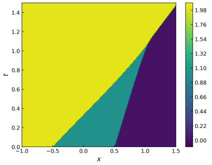
(f) $t = 0.75$

Figure: Comparison results between the reference solution and the predicted solutions by IFNN and PINN

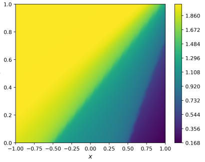
IC: $\phi(x) = \begin{cases} 2, & x \leqslant -1/2, \\ 1, & -1/2 < x \leq 1/2, , \\ 0, & x > 1/2 \end{cases}$ Neumann BC: $u'(-1,t) = u'(3/2,t) = 0$



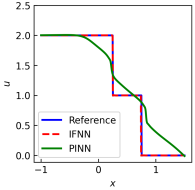
(a) reference solution



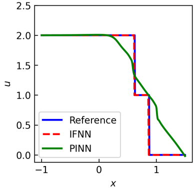
(b) predict solution by IFNN



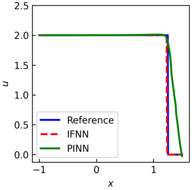
(c) predict solution by PINN



(d) $t = 0.25$



(e) $t = 0.5$

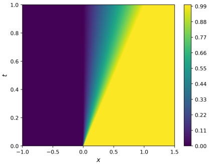


(f) $t = 0.75$

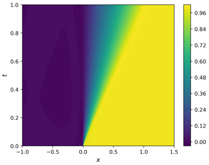
Figure: Comparison results between the reference solution and the predicted solutions by IFNN and PINN

Numerical Experiments - 1D inviscid Burgers' equation with rarefraction wave

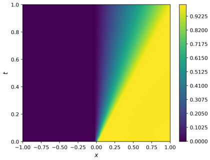
IC: $\phi(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$, Dirichlet BC: $u(-1,t) = 0$



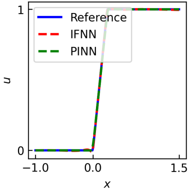
(a) reference solution



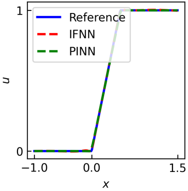
(b) predict solution by IFNN



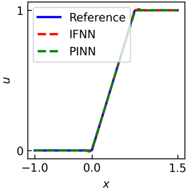
(c) predict solution by PINN



(d) $t = 0.25$



(e) $t = 0.5$

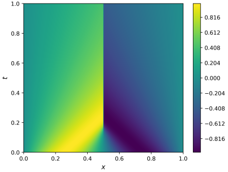


(f) $t = 0.75$

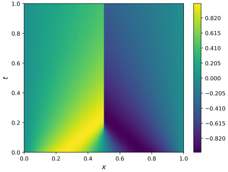
Figure: Comparison results between the reference solution and the predicted solutions by IFNN and PINN

Numerical Experiments - 1D inviscid Burgers' equation with sinusoidal initial data

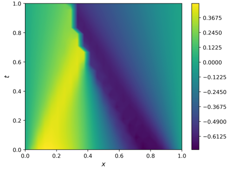
IC: $\phi(x) = \sin(2\pi x)$, Periodic BC: $u(0,t) = u(1,t)$



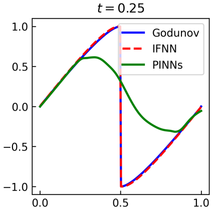
(a) reference solution



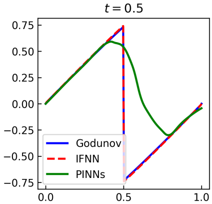
(b) predicted solution by IFNN



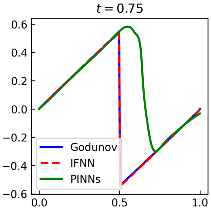
(c) predicted solution by PINN



(d) $t = 0.25$

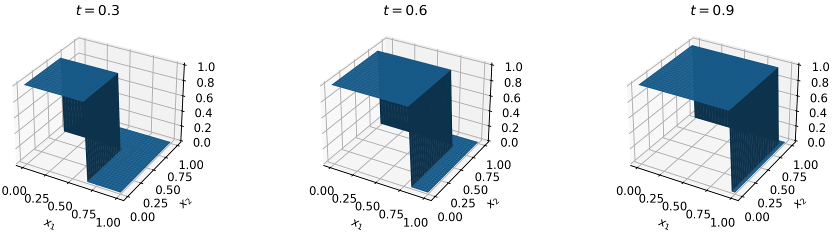


(e) $t = 0.5$

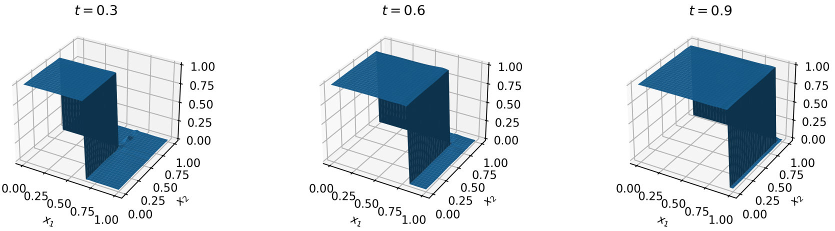


(f) $t = 0.75$

Figure: Comparison results between reference and predicted solutions by IFNN and PINN



(a) Reference solution



(b) Predicted solution by IFNN

Figure: Comparison results between the reference solution and the predicted solution by IFNN

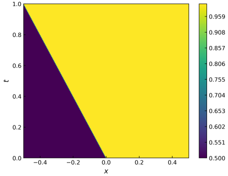
LWR model

By choosing $d = 1$ and $f(u) = u(1-u)$ in (1) we get the Lighthill-Whitham-Richards (LWR) model

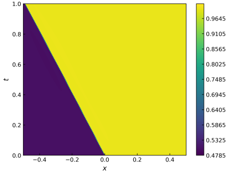
$$\begin{cases} u_t + [u(1-u)]_x = 0, & x \in \mathbb{R}, t > 0, \\ u(x,0) = \phi(x), & x \in \mathbb{R}, \end{cases} \tag{9}$$

where u represents the density of cars on a road and $u \in [0,1]$. When $u = 0$, there are no car on the road, and the road is completely full when $u = 1$.

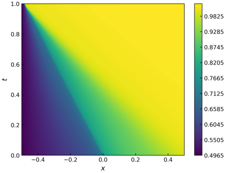
IC: $\phi(x) = \begin{cases} 1/2, & x < 0, \\ 1, & x > 0, \end{cases}$, Dirichlet BC: $u(-1/2, t) = 1/2, \quad t \in (0, 1]$



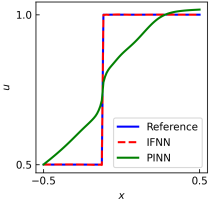
(a) Reference solution



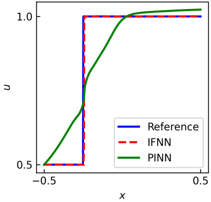
(b) Predicted solution by IFNN



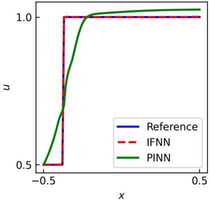
(c) Predicted solution by PINN



(d) $t = 0.25$



(e) $t = 0.5$

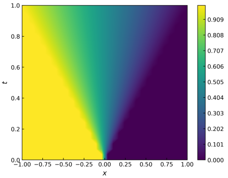


(f) $t = 0.75$

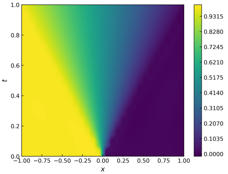
Figure: Comparison results between reference solution and predicted solutions by IFNN and PINN

Numerical Experiments -LWR model for the traffic flow problem(traffic light turning green)

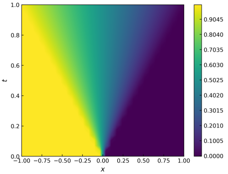
IC: $\phi(x) = \begin{cases} 1, & x < 0, \\ 0, & x > 0, \end{cases}$, Dirichlet BC: $u(-1, t) = 1$



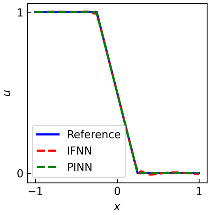
(a) Reference solution



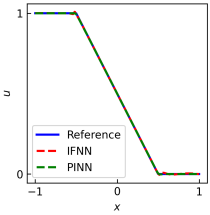
(b) Predicted solution by IFNN



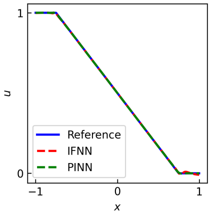
(c) Predicted solution by PINN



(d) $t = 0.25$



(e) $t = 0.5$



(f) $t = 0.75$

Figure: Comparison results between reference solution and predicted solutions by IFNN and PINN

Conclusion remarks

Conclusion

- A fully-connected neural network with skip connection, "IFNN", is proposed for solving the scalar conservation laws.
- The essential difference between our IFNN and PINN lies the choice of the loss function. IFNN takes the implicit form of the solution to formulate one of the essential terms for the loss function while PINN directly uses the residual of the original PDE.
- Extensive numerical experiments in 1D and 2D show that our IFNN is superior to PINN in capturing shock waves while their performance are comparable for the continuous solution cases.

Pros and Cons

- Pros

  the training of IFNN is much easier than that of PINN since it need not to use automatic differentiation for calculations of differential operators
- Cons

  IFNN requires the target PDEs to have the specific implicit form for their solutions, thus the scope of its feasibility may not be as wide as PINN

Thanks for your attention!