Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2021-0257

Vol. **32**, No. 2, pp. 299-335
August 2022

MOD-Net: A Machine Learning Approach via Model-Operator-Data Network for Solving PDEs

Lulu Zhang[1], Tao Luo[2,3], Yaoyu Zhang[1,3,4], Weinan E[5,6], Zhi-Qin John Xu[1,3,*] and Zheng Ma[2,3,*]

[1] *Institute of Natural Sciences and School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai, 200240, China.*
[2] *School of Mathematical Sciences and Institute of Natural Sciences, Shanghai Jiao Tong University, CMA-Shanghai, Shanghai, 200240, China.*
[3] *MOE-LSC and Qing Yuan Research Institute, Shanghai Jiao Tong University, Shanghai, 200240, China.*
[4] *Shanghai Center for Brain Science and Brain-Inspired Technology, Shanghai, 200031, China.*
[5] *School of Mathematical Sciences, Peking University, Beijing, 100871, China.*
[6] *AI for Science Institute, Beijing, 100080, China.*

Received 28 December 2021; Accepted (in revised version) 10 June 2022

Abstract. In this paper, we propose a machine learning approach via model-operator-data network (MOD-Net) for solving PDEs. A MOD-Net is driven by a model to solve PDEs based on operator representation with regularization from data. For linear PDEs, we use a DNN to parameterize the Green's function and obtain the neural operator to approximate the solution according to the Green's method. To train the DNN, the empirical risk consists of the mean squared loss with the least square formulation or the variational formulation of the governing equation and boundary conditions. For complicated problems, the empirical risk also includes a few labels, which are computed on coarse grid points with cheap computation cost and significantly improves the model accuracy. Intuitively, the labeled dataset works as a regularization in addition to the model constraints. The MOD-Net solves a family of PDEs rather than a specific one and is much more efficient than original neural operator because few expensive labels are required. We numerically show MOD-Net is very efficient in solving Poisson equation and one-dimensional radiative transfer equation. For nonlinear PDEs, the nonlinear MOD-Net can be similarly used as an ansatz for solving nonlinear PDEs, exemplified by solving several nonlinear PDE problems, such as the Burgers equation.

AMS subject classifications: 35C15, 35J05, 35Q20, 35Q49, 45K05

Key words: Deep neural network, radiative transfer equation, Green's method, neural operator.

*Corresponding author. *Email addresses:* xuzhiqin@sjtu.edu.cn (Z.-Q. J. Xu), zhengma@sjtu.edu.cn (Z. Ma)

1 Introduction

Nowadays, using deep neural networks (DNNs) to solve PDEs attracts more and more attention [4–7, 9, 10, 12–20]. Here we review three DNN approaches for solving PDEs.

The first approach is to parameterize the solution by a DNN and use the mean square of the residual of the equation [3, 19] or the variational forms [7, 15] as risk or loss, by minimizing which the DNN output satisfies PDE. A comprehensive overview can be found in [6]. This parameterization approach can solve very high-dimensional PDEs and does not require any labels. However, it only solves a specific PDE during each training trial, that is, if the PDE setup changes, such as the source terms, the boundary conditions or other parameters in the PDE, we have to train a new DNN. An important characteristic of the parameterization approach is slow learning of the high frequency part as indicated by the frequency principle [18,24,25]. To overcome the curse of high frequency, a series of multiscale approaches are proposed [1, 12, 16, 21, 23]. The second approach uses DNN to learn the mapping from the source term to the solution [8]. In this approach, the source function and the solution are sampled at fixed grid points as two vectors. Then the vector of the source function is fed into the DNN to predict the vector of the solution function. The advantage of this mapping approach is that the DNN solves the PDE for any source function, thus it can be very convenient in application. However, the mapping approach can only evaluate the solution at fixed points. DeepOnet [17] is proposed that the source function is still fed into the network on fixed grid points but the output can be evaluated on any points by adding on extra inputs of the points to the network. Such approach requires very large sample points, which is often computational inefficient or intractable, especially when dealing with high-dimensional PDEs or complicated integro-differential equations, such as Boltzmann equation and radiative transfer equation (RTE). The third approach is called neural operator [13, 14], which represents the solution based on the form similar to the idea of the Green's function and the DNN is used to parametrize the Green's function. The neural operator solves a type of PDEs but not a specific PDE and can be evaluated at any time or spatial points. Training of the neural operator is to minimize the difference between the learned solution and the true solution at randomly sampled points. Therefore, the neural operator is a data-driven method and requires a large amount of labels, a similar difficulty to the mapping approach.

In this work, we propose a machine learning approach via model-operator-data network (MOD-Net) for solving PDEs. The MOD-Net has advantages including: (i) obtaining a functional representation of the solution which allows evaluating the solution at any points; (ii) requiring none or few labels numerically computed by a traditional scheme on coarse grid points with cheap computation; (iii) solving a family of PDEs but not a specific PDE. The three key components of MOD-Net are illustrated as follows.

Model driven. MOD-Net is driven by the physical model to avoid using too much expensive labeled data. That is, the empirical risk, i.e., training loss requires the solution satisfying the constraints of the governing PDE or equivalent forms and boundary condi-

tions. To realize the model constraint, one can use various methods, such as minimizing the mean square of the residual of the governing PDE and boundary conditions (e.g., physics-informed neural network [3, 19]), or minimizing the variational form of the governing PDE and boundary conditions (e.g., Deep-Ritz method [7]).

Operator representation. Similar to the neural operator [13,14], the MOD-Net represents the solution operator of a PDE, i.e, mapping from source terms, boundary conditions, or parameters to the solution. In this work, DNN is used to parameterize the Green's function, however, it is not restricted to use Green's function and can be generalized to other architectures. This operator representation utilizes the invariant characteristic of Green's function in solving PDE, thus, might be more efficient than an end-to-end representation by parameterizing solution with a DNN directly.

Data regularization. In MOD-Net, we find that in complicated problems, with only the model constraints, the solution is often very inaccurate even when the empirical risk is reasonably small. For example, the radiative/linear transport equations only have a hypercoercive integro-differential operator instead of a nice coercive operator like common elliptic equations. This degeneracy property makes the velocity space may have bad regularity provided some singular coefficients in the equation thus leads to the existence of many "weak solutions" to choose from. To overcome this problem, we add a regularization term by minimizing the difference between the MOD-Net prediction and a few labels numerically computed by a traditional scheme on coarse grid points with cheap computation cost. Note that with only the small amount of labeled data, the MOD-Net cannot be well trained either. Therefore, the effect of the labeled data in MOD-Net is different from supervised learning, in which a DNN training often requires a large amount of accurate labeled data. Intuitively, the effect of data in MOD-Net works as an regularization similar to various regularization terms in traditional optimization problems.

We first apply MOD-Net to solve simple Poisson equations, in which we show that without labels, MOD-Net with the mean square loss or the variational loss of PDE, i.e., governing equation and boundary conditions, can learn the Poisson equation well. We further apply MOD-Net to a class of equations controlled by parameters, which can be regarded as PDEs with uncertainty or a simplification of complicated control problem. For these equations, we can not train the MOD-Net well with only the physical information, however, with the model information and a few labels, we can well train MOD-Net. The data regularization also significantly improves the model accuracy for the RTE [2, 11], which is important in real applications, such as simulation of nuclear reactor, optical tomography and radiation therapy. Besides for these linear PDEs, we also apply nonlinear MOD-Net to the one-dimensional Burgers equation and two-dimensional nonlinear equations. In these two cases, we use no labeled data and only utilize the information of the PDE and we can also well train the nonlinear MOD-Net.

The rest of the paper is organized as follows. In Section 2, we will give a brief introduction of DNNs. Section 3 will present MOD-Net structures. Section 4 will show

the numerical results for Poisson equations. In Section 5, we show the numerical experiments for constructed toy equations. In Section 6, we show numerical experiments for one-dimensional RTE. In Section 7, we show numerical results for one-dimensional Burgers equation. Section 8 will show the numerical experiments for two-dimensional nonlinear equation. Finally, Section 9 gives a conclusion and some discussions for future work.

2 Preliminary: Deep neural networks

We introduce the following conventional notations for DNNs[†]. An L-layer neural network is defined recursively as,

$$\begin{aligned} &\boldsymbol{f}_{\boldsymbol{\theta}}^{[0]}(\boldsymbol{x})=\boldsymbol{x}, \\ &\boldsymbol{f}_{\boldsymbol{\theta}}^{[l]}(\boldsymbol{x})=\sigma\circ(\boldsymbol{W}^{[l-1]}\boldsymbol{f}_{\boldsymbol{\theta}}^{[l-1]}(\boldsymbol{x})+\boldsymbol{b}^{[l-1]}), \quad 1\le l\le L-1, \\ &\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})=\boldsymbol{f}_{\boldsymbol{\theta}}^{[L]}(\boldsymbol{x})=\boldsymbol{W}^{[L-1]}\boldsymbol{f}_{\boldsymbol{\theta}}^{[L-1]}(\boldsymbol{x})+\boldsymbol{b}^{[L-1]}, \end{aligned} \tag{2.1}$$

where $\boldsymbol{W}^{[l]}\in\mathbb{R}^{m_{l+1}\times m_l}$, $\boldsymbol{b}^{[l]}=\mathbb{R}^{m_{l+1}}$, $m_0=d_{\text{in}}=d$ is the input dimension, $m_L=d_{\text{o}}$ is the output dimension, σ is a scalar function and "$\circ$" means entry-wise operation. We denote the set of parameters by $\boldsymbol{\theta}$.

A loss function $\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}),\boldsymbol{y})$ measures the difference between a prediction and a true label. The empirical risk, also known as the training loss for a set $S=\{(\boldsymbol{x}_i,\boldsymbol{y}_i)\}_{i=1}^n$ is denoted by $R_S(\boldsymbol{\theta})$,

$$R_S(\boldsymbol{\theta})=\frac{1}{n}\sum_{i=1}^{n}\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i),\boldsymbol{y}_i). \tag{2.2}$$

More generally, the empirical risk can be defined without labels. Two common empirical risks used for solving PDEs are least square loss (e.g., physics-informed neural network [3, 19]) and variational loss (e.g., Deep-Ritz method [7]). For example, if we want to use DNN $f_{\boldsymbol{\theta}}$ to learn the solution $u(\boldsymbol{x})$ of an equation such as $\mathcal{L}[u](\boldsymbol{x})=\boldsymbol{0}$ for $\boldsymbol{x}\in\Omega$, the empirical risk with least square loss for this equation can be defined by

$$R_S(\boldsymbol{\theta})=\frac{1}{n}\sum_{i=1}^{n}\|\mathcal{L}[f_{\boldsymbol{\theta}}](\boldsymbol{x}_i)\|_2^2, \tag{2.3}$$

where the dataset $\{\boldsymbol{x}_i\}_{i=1}^n$ is randomly sampled from Ω at each iteration step. For some problems, we can use the variational form of the equation to define the loss. More precisely if the solution is the minimizer of the functional $I[u](x)$ whose density denoted by $\ell[u](x)$, then the variational loss can be defined by

$$R_S(\boldsymbol{\theta})=\frac{1}{n}\sum_{i=1}^{n}\ell[f_{\boldsymbol{\theta}}](\boldsymbol{x}_i), \tag{2.4}$$

[†]BAAI.2020. Suggested Notation for Machine Learning. https://github.com/mazhengcn/suggested-notation-for-machine-learning.

where the dataset $\{\boldsymbol{x}_i\}_{i=1}^n$ is randomly sampled from Ω at each iteration step.

The empirical risk can also be defined by the weighted summation of the empirical risk of the labeled data and the empirical risk of the equation. If more constraints needed in the problem, such as boundary conditions, the empirical risk can be similarly adopted.

3 Model-operator-data network (MOD-Net)

Our goal is to solve the following PDE efficiently and accurately,

$$\begin{cases} \mathcal{L}[u](\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = \phi(\boldsymbol{x}), & \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{3.1}$$

where $\mathcal{L}$ is the operator that can be a usual differential operator or even integro-differential operator. Our basic idea is to use a DNN to learn the operator $\mathcal{G}:(\phi,g)\mapsto u$, i.e., for each given boundary condition ϕ and source term g, there is $\mathcal{G}(\phi,g)=u$.

For a linear PDE, Green's function can help us obtain the solution of PDE due to the superposition principle. With the Green's function method, we have the following representation of the solution of PDE (3.1),

$$u(\boldsymbol{x};\phi,g) = \int_\Omega G_1(\boldsymbol{x},\boldsymbol{x}')g(\boldsymbol{x}')\,\mathrm{d}\boldsymbol{x}' + \int_{\partial\Omega} G_2(\boldsymbol{x},\boldsymbol{x}')\phi(\boldsymbol{x}')\,\mathrm{d}\boldsymbol{x}'. \tag{3.2}$$

where for any fixed $\boldsymbol{x}'\in\Omega$, $G_1(\boldsymbol{x},\boldsymbol{x}')$ is the solution of the following equation,

$$\begin{cases} \mathcal{L}[G_1](\boldsymbol{x}) = \delta(\boldsymbol{x}-\boldsymbol{x}'), & \boldsymbol{x} \in \Omega, \\ G_1(\boldsymbol{x},\boldsymbol{x}') = 0, & \boldsymbol{x} \in \partial\Omega, \end{cases}$$

and for any fixed $\boldsymbol{x}'\in\partial\Omega$, $G_2(\boldsymbol{x},\boldsymbol{x}')$ is the solution of the following equation,

$$\begin{cases} \mathcal{L}[G_2](\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Omega, \\ G_2(\boldsymbol{x},\boldsymbol{x}') = \delta(\boldsymbol{x}-\boldsymbol{x}'), & \boldsymbol{x} \in \partial\Omega. \end{cases}$$

For the nonlinear PDE, we can extend the Green's function method for the nonlinear case and use the following representation,

$$u(\boldsymbol{x};\phi,g) = F\left(\int_\Omega G_1(\boldsymbol{x},\boldsymbol{x}')g(\boldsymbol{x}')\,\mathrm{d}\boldsymbol{x}' + \int_{\partial\Omega} G_2(\boldsymbol{x},\boldsymbol{x}')\phi(\boldsymbol{x}')\,\mathrm{d}\boldsymbol{x}'\right), \tag{3.3}$$

where $F(\boldsymbol{x})$ is a nonlinear function and is represented by DNN $F_{\boldsymbol{\theta}}(\boldsymbol{x})$ in this work.

However, it is difficult to obtain the analytical formula of the operators G_1 and G_2. In the following, we consider using DNN to represent operators G_1 and G_2, i.e., a DNN $G_{\boldsymbol{\theta}_1}(\boldsymbol{x},\boldsymbol{x}')$ is trained to represent $G_1(\boldsymbol{x},\boldsymbol{x}')$, similarly, another DNN $G_{\boldsymbol{\theta}_2}(\boldsymbol{x},\boldsymbol{x}')$ is used for $G_2(\boldsymbol{x},\boldsymbol{x}')$. By implementing $G_{\boldsymbol{\theta}_1}(\boldsymbol{x},\boldsymbol{x}')$ and $G_{\boldsymbol{\theta}_2}(\boldsymbol{x},\boldsymbol{x}')$ into Eq. (3.2) and Eq. (3.3), we obtain

a DNN representation for $u(\boldsymbol{x};\phi,g)$ as $u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(\boldsymbol{x};\phi,g)$. In application, the integration in Eq. (3.2) and Eq. (3.3) is realized by discrete numerical schemes. For example, we consider Monte-Carlo algorithm where we uniformly sample a set $S_{G,\Omega}$ from Ω and a set $S_{G,\partial\Omega}$ from $\partial\Omega$, then, for the linear PDE,

$$u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(\boldsymbol{x};\phi,g)=\frac{|\Omega|}{|S_{G,\Omega}|}\sum_{\boldsymbol{x}'\in S_{G,\Omega}}G_{\boldsymbol{\theta}_1}(\boldsymbol{x},\boldsymbol{x}')g(\boldsymbol{x}')+\frac{|\partial\Omega|}{|S_{G,\partial\Omega}|}\sum_{\boldsymbol{x}'\in S_{G,\partial\Omega}}G_{\boldsymbol{\theta}_2}(\boldsymbol{x},\boldsymbol{x}')\phi(\boldsymbol{x}'),\tag{3.4}$$

and for the nonlinear PDE,

$$u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(\boldsymbol{x};\phi,g)=F_{\boldsymbol{\theta}}\Big(\frac{|\Omega|}{|S_{G,\Omega}|}\sum_{\boldsymbol{x}'\in S_{G,\Omega}}G_{\boldsymbol{\theta}_1}(\boldsymbol{x},\boldsymbol{x}')g(\boldsymbol{x}')+\frac{|\partial\Omega|}{|S_{G,\partial\Omega}|}\sum_{\boldsymbol{x}'\in S_{G,\partial\Omega}}G_{\boldsymbol{\theta}_2}(\boldsymbol{x},\boldsymbol{x}')\phi(\boldsymbol{x}')\Big).\tag{3.5}$$

To train the neural networks, we would utilize the information of PDE, i.e., governing equation and boundary conditions, and a few data $S^{u,k}=\{\boldsymbol{x}_i,u^k(\boldsymbol{x}_i)\}_{i\in[n_k]}$ for each $\{\phi^k,g^k\}$, $k=1,2,\cdots,K$, where K is the total number of examples/observations and $u^k(\cdot)=u(\cdot;\phi^k,g^k)$. Note that $S^{u,k}$ can be numerically solved by traditional schemes on coarse grid points, which is not computationally expensive or even obtained from experiment observations. To utilize the constraint of governing PDE, for each k, we uniformly sample a set of data from Ω, i.e., $S^{\Omega,k}$. To utilize the information of boundary constraint, for each k, we uniformly sample a set of data from $\partial\Omega$, i.e., $S^{\partial\Omega,k}$. Then, we train the neural networks by minimizing the empirical risk defined as follows,

$$\begin{aligned}R_S=\frac{1}{K}\sum_{k\in[K]}\Big(&\lambda_1\frac{1}{|S^{\Omega,k}|}\sum_{\boldsymbol{x}\in S^{\Omega,k}}\|\mathcal{L}[u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(\boldsymbol{x};\phi^k,g^k)](\boldsymbol{x})-g^k(\boldsymbol{x})\|_2^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega,k}|}\sum_{\boldsymbol{x}\in S^{\partial\Omega,k}}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(\boldsymbol{x};\phi^k,g^k)-\phi^k(\boldsymbol{x})\|_2^2\\&+\lambda_3\frac{1}{n_k}\sum_{i\in[n_k]}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(\boldsymbol{x}_i;\phi^k,g^k)-u^k(\boldsymbol{x}_i)\|_2^2\Big),\end{aligned}\tag{3.6}$$

where $\lambda_1,\lambda_2,\lambda_3$ are hyperparameters used to tune the weight of each part in the total risk. The Schematic of MOD-Net approach is shown in Fig. 1.

Here we remark that the least square loss is not crucial, we can use other loss such as variational loss, see Example 2. Also the labeled data is not restricted the true solution, we can use other data as regularization term, such as the macroscopic quantities, e.g., the density function $\rho(x_i)$ in the RTE, which is a moment of the solution. An important advantage of our proposed MOD-Net method is that we take advantage of the PDE constraint and use cheap not-so-accurate labeled data. For convenience, the notations are listed in Table 1.

In real applications, depending on the problems, if the boundary condition is zero, such as the Poisson case in Section 4, G_2 is ignored, and if the source term is zero, such as the RTE case in Section 6, G_1 is ignored. For one-dimensional case, the integration

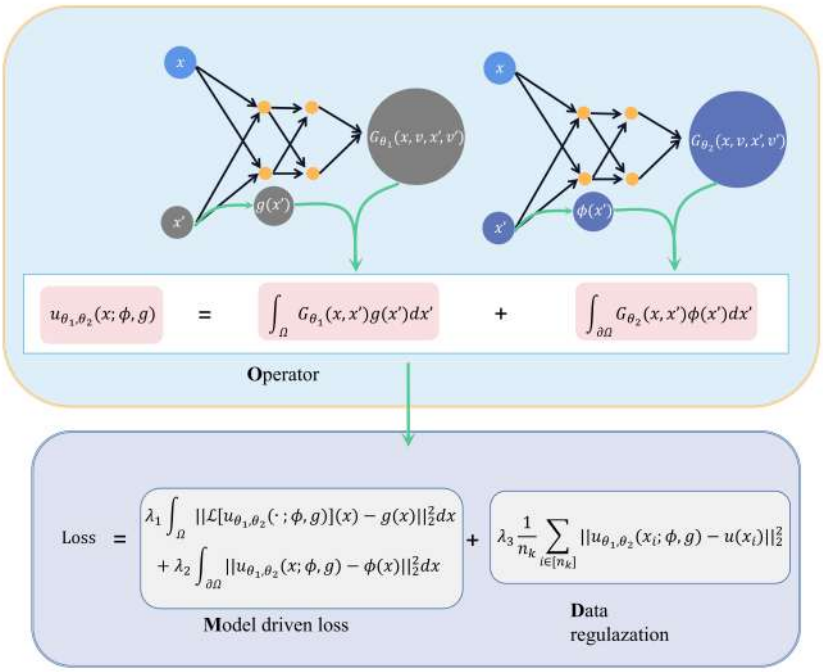


Figure 1: Schematic of MOD-Net approach. In MOD-Net approach, we use the DNN G_{θ_1} and G_{θ_2} to parameter the Green's function G_1 and G_2, respectively. And according to the Green's formula, we obtain u_{θ_1,θ_2} which can approximate solution operator. Then we train these DNNs utilizing the information of PDE, i.e., governing equation and boundary conditions, and a few data.

over boundary is the summation at two points, therefore, we can use two DNNs to learn $G_2(\boldsymbol{x},\boldsymbol{x}')|_{\boldsymbol{x}'=\boldsymbol{x}_L}$ and $G_2(\boldsymbol{x},\boldsymbol{x}')|_{\boldsymbol{x}'=\boldsymbol{x}_R}$, respectively, where $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$ are boundary points. The input of the two DNNs are lower dimensional due to the fixation of $\boldsymbol{x}'$. The amount of the labeled data required also depends on the problem. For simple problems, such as Poisson problem, we use no labeled data, but for complicated constructed toy example or RTE problem, we use a few labeled data.

4 Numerical experiments: 2D Poisson equation

We consider the Poisson equations in 2D,

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \boldsymbol{x}\in\Omega, \\ u(\boldsymbol{x}) &= 0, \quad \boldsymbol{x}\in\partial\Omega. \end{aligned} \tag{4.1}$$

where the source function $g(x,y)=-a(x^2-x+y^2-y)$, i.e.,

$$\begin{aligned} -(\partial_{xx}u+\partial_{yy}u) &= -a(x^2-x+y^2-y), \quad (x,y)\in\Omega, \\ u &= 0, \quad (x,y)\in\partial\Omega, \end{aligned} \tag{4.2}$$

Table 1: Notation.

$\mathcal{L}$	PDE operator
g	source term in PDE
ϕ	boundary value in PDE
$\mathcal{G}$	operator to be learned, general definition is $\mathcal{G}:(\phi,\sigma,g)\mapsto u$
$[n]$	index set $\{1,2,...,n\}$
$S_{G_x},S_{G_y},S^+_{G_v},S^-_{G_v}$	the set of integration points in solution's representation
S_v	the set of integration points in v direction
ω	integration coefficients
S^Ω	a set of data uniformly sampled from Ω
$S^{\partial\Omega}$	a set of data uniformly sampled from $\partial\Omega$
$S^{u,k}=\{\boldsymbol{x}_i,u^k(\boldsymbol{x}_i)\}_{i=1}^{n_k}$	labeled data set for kth PDE, where $u^k(\cdot)$ is solution of kth PDE
$F_{\boldsymbol{\theta}}(\boldsymbol{x}),G_{\boldsymbol{\theta}}(\boldsymbol{x}),u_{\boldsymbol{\theta}}(\boldsymbol{x})$	DNN

where $\Omega=[0,1]^2$ and the constant a controls the source term. Obviously, the analytical solution is $u(x,y;g)=\frac{a}{2}x(x-1)y(y-1)$.

4.1 Use DNN to fit Green's function

For Poisson equation, which is a linear PDE, using the Green's function method, the solution of (4.1) can be represented by

$$u(\boldsymbol{x};g)=\int_\Omega G(\boldsymbol{x},\boldsymbol{x}')g(\boldsymbol{x}')\,\mathrm{d}\boldsymbol{x}', \tag{4.3}$$

where for any $x'\in\Omega$, the Green's function $G(\boldsymbol{x},\boldsymbol{x}')$ is the solution of following problem,

$$\begin{aligned} -\Delta G(\boldsymbol{x},\boldsymbol{x}')&=\delta(\boldsymbol{x}-\boldsymbol{x}'), \quad \boldsymbol{x}\in\Omega,\\ G(\boldsymbol{x},\boldsymbol{x}')&=0, \quad \boldsymbol{x}\in\partial\Omega. \end{aligned} \tag{4.4}$$

In the considered 2D case, $\Omega=[0,1]^2$ and $g(x,y)=-a(x^2-x+y^2-y)$, we have

$$u(x,y;g)=\int_0^1\int_0^1 G(x,y,x',y')g(x',y')\,\mathrm{d}x'\,\mathrm{d}y'. \tag{4.5}$$

For demonstration, although we can obtain the analytical form of the Green's function, we use a DNN of hidden layer size 128-128-128-128 $G_{\boldsymbol{\theta}}(x,y,x',y')$ to fit the Green's function $G(x,y,x',y')$.

When we calculate the integral, it is impossible to integrate it analytically. In practice, we often use the numerical integration. We use the Gauss-Legendre quadrature. Then we can represent neural operator $u_{\boldsymbol{\theta}}(x,y;g)$ with Green's function DNN $G_{\boldsymbol{\theta}}(x,y,x',y')$, that is,

$$u_{\boldsymbol{\theta}}(x,y;g)=\sum_{x'\in S_{G_x}}\sum_{y'\in S_{G_y}}\omega_{x'}\omega_{y'}G_{\boldsymbol{\theta}}(x,y,x',y')g(x',y'), \tag{4.6}$$

where $S_{G_x} \subset [0,1]$, $S_{G_y} \subset [0,1]$ consist fixed integration points, determined by 1D Gauss-Legendre quadrature and $\omega_{x'}$, $\omega_{y'}$ are corresponding coefficients. Note that, using the deterministic quadrature points is not crucial here and only for illustration purpose. In fact, for high-dimensional PDEs we need to use the Monte Carlo method to overcome the difficulty of high-dimensional integration, more specifically, we represent neural operator $u_{\boldsymbol{\theta}}(x,y;g)$ with Green's function DNN $G_{\boldsymbol{\theta}}(x,y,x',y')$ as follows,

$$u_{\boldsymbol{\theta}}(x,y;g) = \frac{|\Omega|}{N}\sum_{i=1}^{N} G_{\boldsymbol{\theta}}(x,y,x_i',y_i')g(x_i',y_i'),$$

where $S_{G,\Omega} = \{(x_i',y_i')\}_{i=1}^N$ are uniformly sampled from Ω. Here we need to emphasize that above sampled points set is different in every training step.

4.2 Empirical risk function

For this toy example, to train the neural networks, we use no labeled data and only utilize the information of PDE, i.e., governing equation and boundary condition, for each g^k, $k=1,2,\cdots,K$. To utilize the constraint of governing equation of PDE, we uniformly sample a set of data from $\Omega=[0,1]\times[0,1]$, i.e., $S^{\Omega,k}$. To utilize the information of boundary constraint, for each k, we uniformly sample a set of data from $\partial\Omega$. Since the boundary $\partial\Omega$ consists of four line segments, i.e., $\partial\Omega = \bigcup_{i=1}^4 \partial\Omega_i$, where $\partial\Omega_1 = \{(0,y)\}_{y\in[0,1]}$, $\partial\Omega_2 = \{(1,y)\}_{y\in[0,1]}$, $\partial\Omega_3 = \{(x,0)\}_{x\in[0,1]}$, $\partial\Omega_4 = \{(x,1)\}_{x\in[0,1]}$, we uniformly sample a set of data from $\partial\Omega_i$ respectively, i.e., $S^{\partial\Omega_i,k}$, $i=1,2,3,4$.

Since we use no labeled data, λ_3 in the general definition (3.6) is set as zero. The empirical risk for this example is as follows,

$$\begin{aligned}
R_S = \frac{1}{K}\sum_{k\in[K]}\Bigg(&\lambda_1\frac{1}{|S^{\Omega,k}|}\sum_{(x,y)\in S^{\Omega,k}}\Big(-\big(\partial_{xx}u_{\boldsymbol{\theta}}(x,y;g^k)+\partial_{yy}u_{\boldsymbol{\theta}}(x,y;g^k)\big)-g^k(x,y)\Big)^2\\
&+\lambda_2\frac{1}{|S^{\partial\Omega_1,k}|}\sum_{(x,y)\in S^{\partial\Omega_1,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\
&+\lambda_2\frac{1}{|S^{\partial\Omega_2,k}|}\sum_{(x,y)\in S^{\partial\Omega_2,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\
&+\lambda_2\frac{1}{|S^{\partial\Omega_3,k}|}\sum_{(x,y)\in S^{\partial\Omega_3,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\
&+\lambda_2\frac{1}{|S^{\partial\Omega_4,k}|}\sum_{(x,y)\in S^{\partial\Omega_4,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\Bigg).
\end{aligned} \tag{4.7}$$

Note that, the least square loss in (3.6) is not crucial. To support this point, we also use

the variational loss used in Deep Ritz Method and the empirical risk is as follows,

$$\begin{aligned}R_S=\frac{1}{K}\sum_{k\in[K]}\Bigg(&\lambda_1\frac{1}{|S^{\Omega,k}|}\sum_{(x,y)\in S^{\Omega,k}}\Big(\frac{1}{2}\big(|\partial_x u_{\boldsymbol{\theta}}(x,y;g^k)|^2+|\partial_y u_{\boldsymbol{\theta}}(x,y;g^k)|^2\big)-g^k(x,y)u_{\boldsymbol{\theta}}(x,y;g^k)\Big)\\&+\lambda_2\frac{1}{|S^{\partial\Omega_1,k}|}\sum_{(x,y)\in S^{\partial\Omega_1,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega_2,k}|}\sum_{(x,y)\in S^{\partial\Omega_2,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega_3,k}|}\sum_{(x,y)\in S^{\partial\Omega_3,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega_4,k}|}\sum_{(x,y)\in S^{\partial\Omega_4,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\Bigg).\end{aligned}\tag{4.8}$$

4.3 Learning process

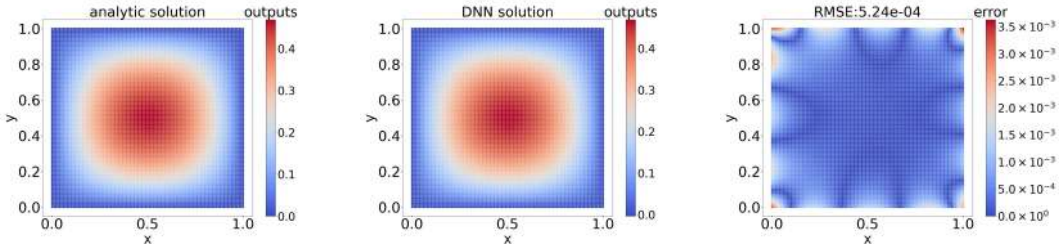
For each training epoch, we first randomly choose source functions $\{g^k\}_{k=1}^K$ and calculate their values on fixed quadrature points(x',y'), where $x'\in S_{G_x}$ and $y'\in S_{G_y}$, or on uniformly samples (x',y'), where $(x',y')\in S_{G,\Omega}$. Second, we randomly sample data and obtain data set $S^{\Omega,k}$,$S^{\partial\Omega_i,k}$, $i=1,2,3,4$. We obtain the dataset $D=\{(x,y,x',y',g^k(x',y'))|(x,y)\in S^{\Omega,k}\cup(\bigcup_{i=1}^4 S^{\partial\Omega_i,k})\}$. In the following, we feed the data into the neural network $G_{\boldsymbol{\theta}}(x,y,x',y')$ and calculate the total risk (4.7) or (4.8) with neural operator $u_{\boldsymbol{\theta}}(x,y;g)$, see Eq. (4.6) or Eq. (4.1). We train neural network $G_{\boldsymbol{\theta}}$ with Adam to minimize the total risk, and finally we obtain a well-trained Green's function DNN $G_{\boldsymbol{\theta}}$, furthermore, according to Eq. (4.6) or Eq. (4.1), we obtain a neural operator $u_{\boldsymbol{\theta}}(x,y;g)$.

4.4 Results

The source function $g(x,y)=-a(x^2-x+y^2-y)$ is determined by a. We then denote source function $g(x,y)=-a(x^2-x+y^2-y)$ by g_a. To illustrate our approach, we train a neural operator mapping from g to the solution of the Poisson equation (4.2).

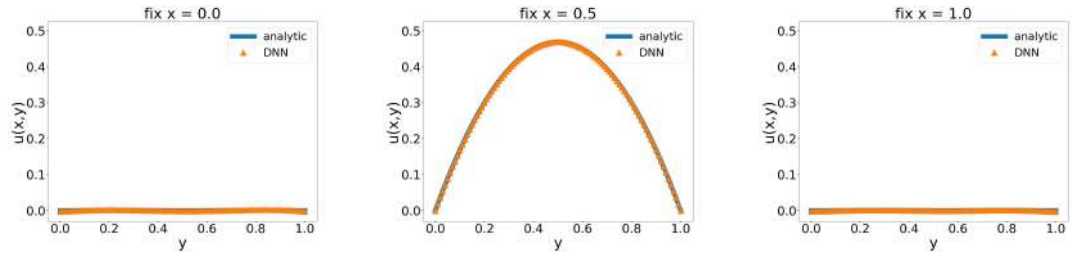
Example 1: MOD-Net for a family of Poisson equations using least square loss. For illustration, we would train the MOD-Net by various source functions and test the MOD-Net with several source functions that are not used for training. In this example, we calculate the empirical risk with least square loss.

Initially, we represent the neural operator using Gauss-Legendre quadrature. During training, for each epoch, we choose a family of source functions, for example, we sample $K=10$ source functions g_a from selected region, that is, we sample control parameter a uniformly from $\{10k\}_{k=1}^{20}$. We set the number of integration points $|S_{G_x}|=10$ and $|S_{G_y}|=10$,



(a) $a=15$ (b) $a=15$ (c) $a=15$

Figure 2: Example 1. Comparison between analytic solution and MOD-Net solution on 101×101 grid points corresponding to source terms determined by $a=15$. (a) Analytic solution. (b) MOD-Net solution. (c) The difference between the analytic solution and MOD-Net solution. Since the error at most points are very small, here we reduce the upper limit of the Color bar to see more information.

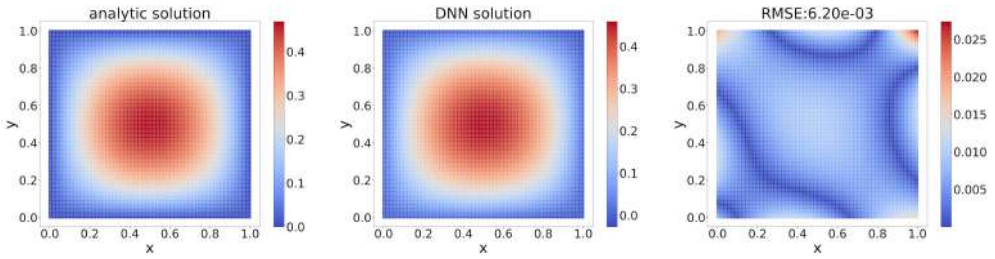


(a) $a=15$, $x=0.0$ (b) $a=15$, $x=0.5$ (c) $a=15$, $x=1.0$

Figure 3: Example 1. Comparison between analytic solution and MOD-Net solution on $x=0,0.5,1$ corresponding to source function determined by test $a=15$.

and for each epoch, we randomly sample 200 points in $[0,1]^2$ and sample 200 points in each line of boundary, respectively. We set $\lambda_1=1$, $\lambda_2=1$ in the empirical risk using least square loss (4.7). We test the performance of this well-trained MOD-Net on $a=15$. For each fixed a, the corresponding exact true solution is $u(x,y)=\frac{a}{2}x(x-1)y(y-1)$. For visualizing the performance of our well-trained MOD-Net, we show the analytic solution and MOD-Net solution on 101×101 equidistributed isometric grid points by color in Fig. 2. To compare these two solutions more intuitively, we calculate the difference between the two solutions at each point and show the error by color, i.e., as the error increases, the color changes from blue to red, in Fig. 2(c). The root of mean square error (RMSE) is $\sim5.2\times10^{-4}$. We also show the solution obtained by MOD-Net and the corresponding analytical solution on fixed $x=0,0.5,1$ for considered test source function. As shown in Fig. 3, the MOD-Net can well predict the analytic solutions.

For illustrating the effectiveness of the Monte Carlo integration in MOD-Net, we uniformly sample 100 points in $\Omega=[0,1]^2$ as integration points denoted by $S_{G,\Omega}$, and all other settings are the same as using Gauss-Legendre quadrature. Note that for illustration and



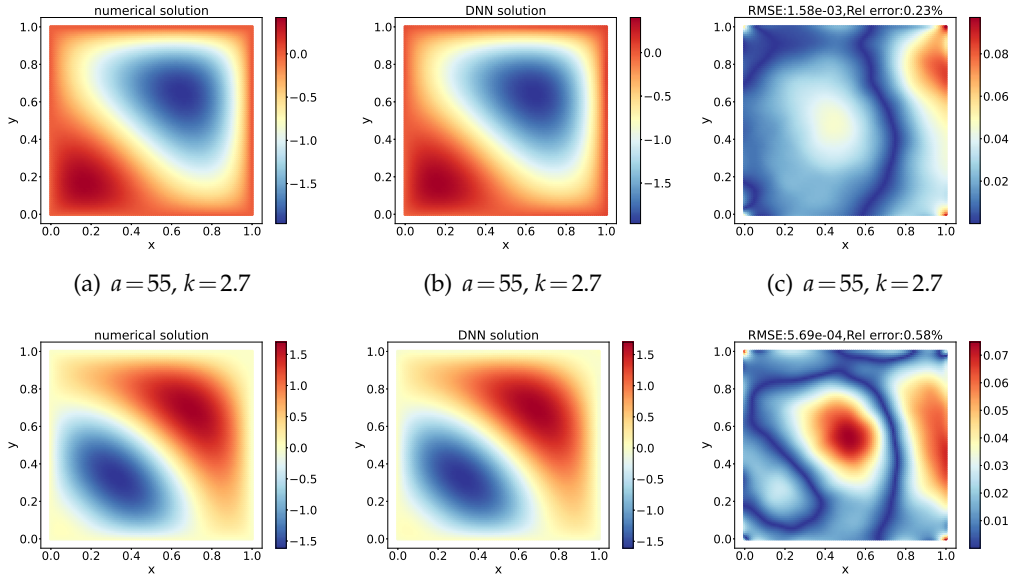
(a) $a=15$ (b) $a=15$ (c) $a=15$

Figure 4: Effect of MOD-Net using Monte Carlo integration. Comparison between analytic solution and MOD-Net solution on 101×101 grid points corresponding to source terms determined by $a=15$. (a) Analytic solution. (b) MOD-Net solution. (c) The difference between the analytic solution and MOD-Net solution.

also to reduce the computational cost, the sampling procedure during training or evaluation is as follows: we equidistantly take 51×51 points in $\Omega=[0,1]^2$ and uniformly sample 100 points in these equidistributed isometric grid points in stead of directly sampling points from $\Omega=[0,1]^2$. As shown in Fig. 4, the MOD-Net also can well predict the analytic solutions with RMSE $\sim 6.2\times10^{-3}$.

We also consider a relatively larger class of source functions g. We set $g(x,y)=a\cdot\cos(k\sqrt{x^2+y^2})$ which is parameterized by a,k. In fact, due to the principle of linear superposition, we can represent the solution corresponding to $g(x,y)=\sum_k a_k\cos(k\sqrt{x^2+y^2})$ by the linear combination of solution in the form of $g(x,y)=a_k\cos(k\sqrt{x^2+y^2})$. To predict the solutions corresponding to source functions $g(x,y)$ with $a\in[10,100]$, $k\in[1,10]$, we use training samples $\{(a_i,k_j)\}_{i,j}=\{10i\}_{i=1}^{10}\times\{1+0.5j\}_{j=0}^{19}$. During training, for each epoch, to sufficiently use these training samples, we randomly shuffle and split these 200 training samples into 20 batches, that is, for each batch, we use $K=10$ training samples. In this example, we represent the neural operator using the Gauss-Legendre quadrature and we set the number of integration points $|S_{G_x}|=10$ and $|S_{G_y}|=10$. For each epoch, we randomly sample 500 points in $\Omega=[0,1]^2$ and sample 500 points in each line of boundary, respectively. We set $\lambda_1=1$, $\lambda_2=100$ in the empirical risk using least square loss (4.8). To see the performance of MOD-Net, we use the spectral method to obtain the numerical solution as a reference. We test the performance of the well-trained MOD-Net on $a=55$, $k=2.7$ and $a=74$, $k=6.3$. To visualize the performance of our well-trained MOD-Net, for each considered test a,k, we show the numerical solution and MOD-Net solution on 101×101 equidistributed isometric grid points by color in Fig. 5. And to compare these two solutions more intuitively, we calculate the difference between the two solutions at each point and show the error by color, i.e., as the error increases, the color changes from blue to red, in Fig. 5(c,f). We see the MOD-Net can predict the solutions well.

Example 2: MOD-Net for a family of Poisson equations using variational loss. In this experiment, we also train the MOD-Net by various source functions. But, note that,



(a) $a=55$, $k=2.7$ (b) $a=55$, $k=2.7$ (c) $a=55$, $k=2.7$

(d) $a=74$, $k=6.3$ (e) $a=74$, $k=6.3$ (f) $a=74$, $k=6.3$

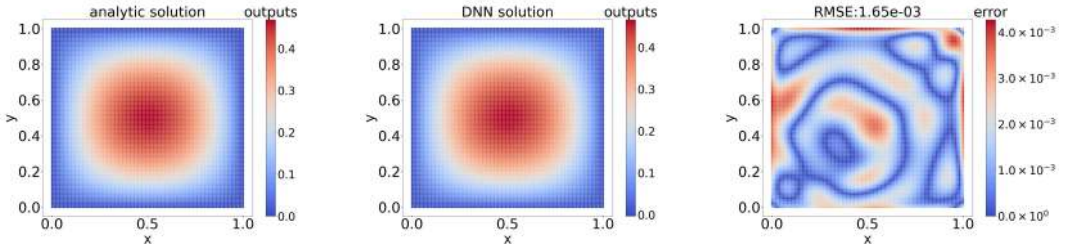
Figure 5: General source function. Comparison between numerical solution and MOD-Net solution corresponding to source function $g(x,y)=a\cdot\cos(k\sqrt{x^2+y^2})$ determined by test $a=55$, $k=2.7$ and $a=74$, $k=6.3$.

in this example, we calculate the empirical risk with variational loss used in Deep Ritz Method.

In training, for each epoch, similarly we set $K=10$ and we sample control parameter a uniformly from $\{10k\}_{k=1}^{20}$. We set the number of integration points $|S_{G_x}|=10$ and $|S_{G_y}|=10$, and for each epoch, we randomly sample 600 points in $[0,1]^2$ and sample 600 points in each line of boundary, respectively. We set $\lambda_1=1$, $\lambda_2=250$ in the empirical risk using least square loss (4.8).

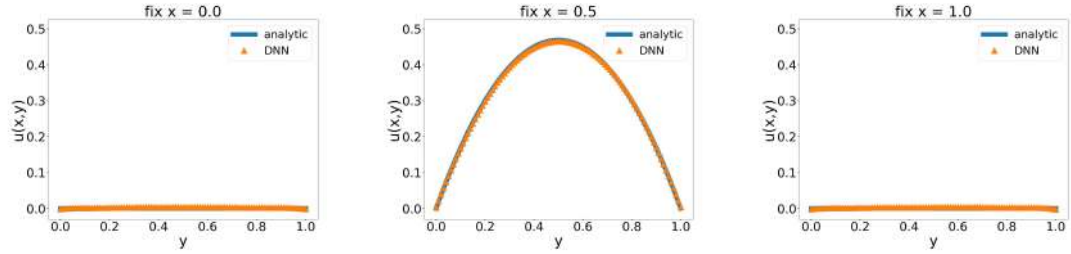
We test the performance of this well-trained MOD-Net on $a=15$. For each fixed a, the corresponding exact true solution is $u(x,y)=\frac{a}{2}x(x-1)y(y-1)$. For visualizing the performance of our well-trained MOD-Net, similarly we show the analytic solution and MOD-Net solution on 101×101 equidistributed isometric grid points by color. To compare these two solutions more intuitively, similarly we calculate the difference between the two solutions at each point and show the error by color in Fig. 6. The root mean square error (RMSE) is $\sim1.6\times10^{-3}$.

For visualizing the performance of our well-trained MOD-Net, we show the solution obtained by MOD-Net and the corresponding analytical solution on fixed $x=0,0.5,1$ for considered test source function. As shown in Fig. 7, the MOD-Net can well predict the analytic solutions.



(a) $a=15$ (b) $a=15$ (c) $a=15$

Figure 6: Example 2. Comparison between analytic solution and MOD-Net solution on 101×101 grid points corresponding to source terms determined by $a=15$. (a) Analytic solution. (b) MOD-Net solution. (c) The difference between the analytic solution and MOD-Net solution.



(a) $a=15$, $x=0.0$ (b) $a=15$, $x=0.5$ (c) $a=15$, $x=1.0$

Figure 7: Example 2. Comparison between analytic solution and MOD-Net solution on $x=0,0.5,1$ corresponding to source function determined by test $a=15$.

5 Numerical experiments: Equation with uncertainty or control variable

In this section, we would use a toy model to show the positive effect of the regularization from a few labels. We consider the following equation,

$$\begin{aligned} \partial_x u(x,y)+a(x,y)u(x,y)&=g(x,y), \quad (x,y)\in\Omega, \\ u(x,y)&=0, \quad (x,y)\in\partial\Omega. \end{aligned} \tag{5.1}$$

where $\Omega=[a,b]\times[c,d]$, the variable y is the auxiliary variable that can be uncertainty variable or the control variable. $a(x,y)$ and $g(x,y)$ are coefficient functions that contain the randomness or the control functions in the control problem. This equation can be regarded as a simplified toy model of a linear ODE with uncertainty or a simplified version of the equation with degeneracy such as the kinetic equations. We use the toy model here to show that data regularization is a crucial ingredients when dealing with this kind of equations.

5.1 Use DNN to fit Green's function

Similarly to Poisson equation, using the Green's function method, the solution of (5.1) can be represented by

$$u(x,y;g)=\int_c^d\int_a^b G(x,y,x',y')g(x',y')\mathrm{d}x'\mathrm{d}y'. \tag{5.2}$$

In this experiment, we use a DNN of hidden layer size 128-128-128-128 $G_{\boldsymbol{\theta}}(x,y,x',y')$ equipped with activation function tanh to fit the Green's function $G(x,y,x',y')$.

When we calculate the integral, we use the Gauss-Legendre quadrature. Then we can represent neural operator $u_{\boldsymbol{\theta}}(x,y;g)$ with Green's function DNN $G_{\boldsymbol{\theta}}(x,y,x',y')$, that is,

$$u_{\boldsymbol{\theta}}(x,y;g)=\sum_{x'\in S_{G_x}}\sum_{y'\in S_{G_y}}\omega_{x'}\omega_{y'}G_{\boldsymbol{\theta}}(x,y,x',y')g(x',y'), \tag{5.3}$$

where $S_{G_x}\subset[a,b]$, $S_{G_y}\subset[c,d]$ consist fixed integration points, determined by 1D Gauss-Legendre quadrature and $\omega_{x'}$, $\omega_{y'}$ are corresponding coefficients.

5.2 Empirical risk function

For this example, to train the neural networks, we utilize the information of PDE, i.e., governing equation and boundary condition and a few data, $S^{u,k}=\{x_i,y_i,u^k(x_i,y_i)\}_{i\in[n_k]}$ for each g^k, $k=1,2,\cdots,K$, where $u^k(\cdot)=u(\cdot;g^k)$. Note that $S^{u,k}$ can be analytically solved on grid points. To utilize the constraint of governing equation of PDE, we uniformly sample a set of data from $\Omega=[a,b]\times[c,d]$, i.e., $S^{\Omega,k}$. To utilize the information of boundary constraint, for each k, we uniformly sample a set of data from $\partial\Omega$. Since the boundary $\partial\Omega$ consists of four line segments, i.e., $\partial\Omega=\bigcup_{i=1}^4\partial\Omega_i$, where $\partial\Omega_1=\{(a,y)\}_{y\in[c,d]}$, $\partial\Omega_2=\{(b,y)\}_{y\in[c,d]}$, $\partial\Omega_3=\{(x,c)\}_{x\in[a,b]}$, $\partial\Omega_4=\{(x,d)\}_{x\in[a,b]}$, we uniformly sample a set of data from $\partial\Omega_i$ respectively, i.e., $S^{\partial\Omega_i,k}$, $i=1,2,3,4$.

The empirical risk for this example is as follows,

$$\begin{aligned}R_S=\frac{1}{K}\sum_{k\in[K]}\Bigg(&\lambda_1\frac{1}{|S^{\Omega,k}|}\sum_{(x,y)\in S^{\Omega,k}}\Big(\partial_x u_{\boldsymbol{\theta}}(x,y;g^k)+a(x,y)u_{\boldsymbol{\theta}}(x,y;g^k)-g^k(x,y)\Big)^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega_1,k}|}\sum_{(x,y)\in S^{\partial\Omega_1,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\end{aligned}$$

$$
\begin{aligned}
&+\lambda_2\frac{1}{|S^{\partial\Omega_2,k}|}\sum_{(x,y)\in S^{\partial\Omega_2,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\
&+\lambda_2\frac{1}{|S^{\partial\Omega_3,k}|}\sum_{(x,y)\in S^{\partial\Omega_3,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\
&+\lambda_2\frac{1}{|S^{\partial\Omega_4,k}|}\sum_{(x,y)\in S^{\partial\Omega_4,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\
&+\lambda_3\frac{1}{n_k}\sum_{i\in[n_k]}|u_{\boldsymbol{\theta}}(x_i,y_i;g^k)-u^k(x_i,y_i)|^2\Big).
\end{aligned}\tag{5.4}
$$

5.3 Learning process

Similarly to Poisson equation, for each training epoch, we first randomly choose source functions $\{g^k\}_{k=1}^K$ and calculate their values on fixed integration points(x',y'), where $x'\in S_{G_x}$ and $y'\in S_{G_y}$. Second, we randomly sample data and obtain data set $S^{\Omega,k}$,$S^{\partial\Omega_i,k}$, $i=1,2,3,4$. We obtain the data set $D=\{(x,y,x',y',g^k(x',y'))|(x,y)\in S^{\Omega,k}\cup(\bigcup_{i=1}^4 S^{\partial\Omega_i,k}),x'\in S_{G_x},y'\in S_{G_y}\}$. In the following, we feed the data into the neural network $G_{\boldsymbol{\theta}}(x,y,x',y')$ and calculate the total risk (5.4) with neural operator $u_{\boldsymbol{\theta}}(x,y;g)$, see Eq. (4.6). Train neural network $G_{\boldsymbol{\theta}}$ with Adam to minimize the total risk, and finally we obtain a well-trained Green's function DNN $G_{\boldsymbol{\theta}}$, furthermore, according to Eq. (4.6), we obtain a neural operator $u_{\boldsymbol{\theta}}(x,y;g)$.

5.4 Results

To demonstrate data regularization, we apply MOD-Net method in the following example.

Example 3: MOD-Net for equation with auxiliary variable. In this example, we consider Eq. (5.1) in which $\Omega=[0,1]\times[1,2]$, $a(x,y)=\frac{\sin(4\pi xy)+2}{y}$ and the source term

$$
\begin{aligned}
&g(x,y)\\
&=\begin{cases}
(y^2-3y+2)\Big(x^2-x+\Big(2x-1+\dfrac{x(x-1)(2+\sin(4\pi xy))}{y}\Big)(x+\cos(\pi y))\Big), & 1\le y<1.5,\\
(y^2-3y+2)\Big(x^2-x+\Big(2x-1+\dfrac{x(x-1)(2+\sin(4\pi xy))}{y}\Big)(x+\sin(2\pi y))\Big), & 1.5\le x\le 2.
\end{cases}
\end{aligned}\tag{5.5}
$$

The graph of source term is shown in Fig. 8. For this equation, the analytical solution

$$
u(x,y)=\begin{cases}
x(x-1)(y-1)(y-2)\Big(x+\cos(\pi y)\Big), & 1\le y<1.5,\\
x(x-1)(y-1)(y-2)\Big(x+\sin(2\pi y)\Big), & 1.5\le x\le 2.
\end{cases}\tag{5.6}
$$

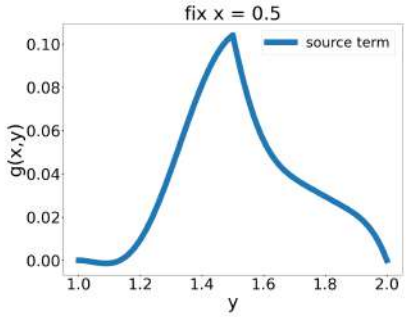
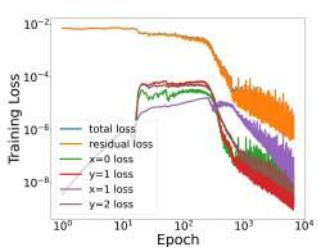
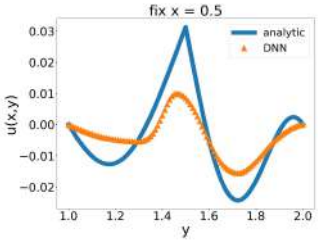


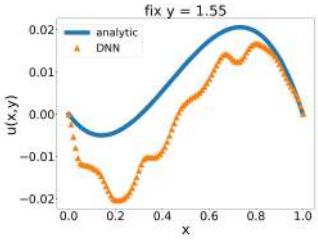
Figure 8: Example 3. Source term $g(x,y)$ on $x=0.5$, i.e., $g(0.5,y)$ vs. y. Remark that $\partial_y g$ is discontinuous.



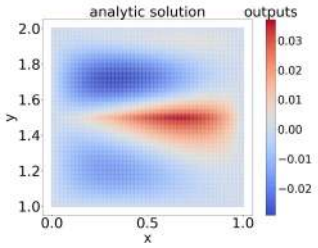
(a) Training loss



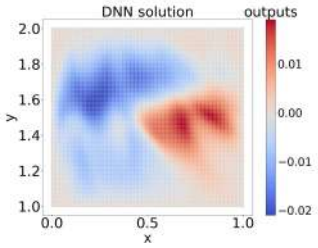
(b) $x=0.5$



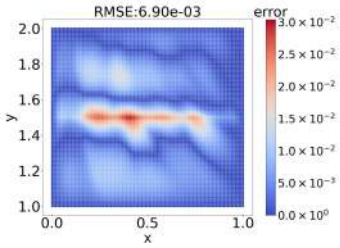
(c) $y=1.55$



(d) Analytic solution



(e) MOD-Net solution

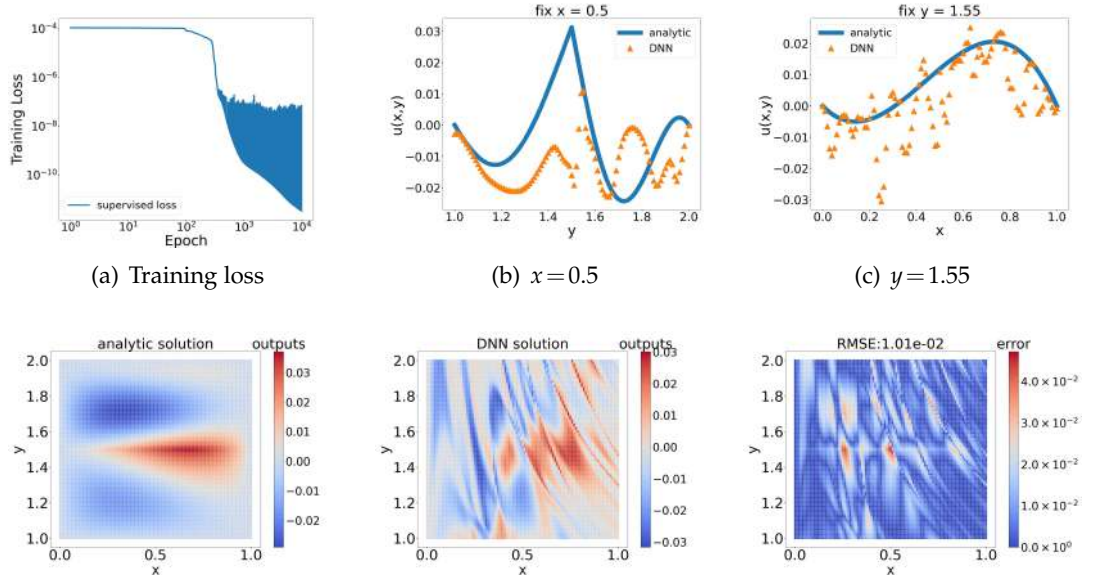


(f) Difference between two solutions

Figure 9: Example 3. Trained by only the PDE,i.e., the governing equation and boundary condition. (a) Training loss. The blue curve is the total risk Eq. (5.4). The other five curves represent five terms in the total risk Eq. (5.4) except the last term, respectively. (b,c) Comparison between analytic solution and MOD-Net solution on $x=0.5$ and $y=1.55$. (d,e,f) Comparison between analytic solution and MOD-Net solution on 101×101 grid points.

Note that the first derivative of solution $u(x,y)$ with respect to x, i.e., $\partial_x u$ is continuous, but $\partial_y u$ is discontinuous. For this example, we train the MOD-Net to approximate the solution of a specific equation with three different loss functions.

First, we set the λ_3 be zero in (5.4), and train the MOD-Net only by the governing equation and boundary conditions. The empirical risk, i.e., training loss is shown in



(a) Training loss (b) $x=0.5$ (c) $y=1.55$

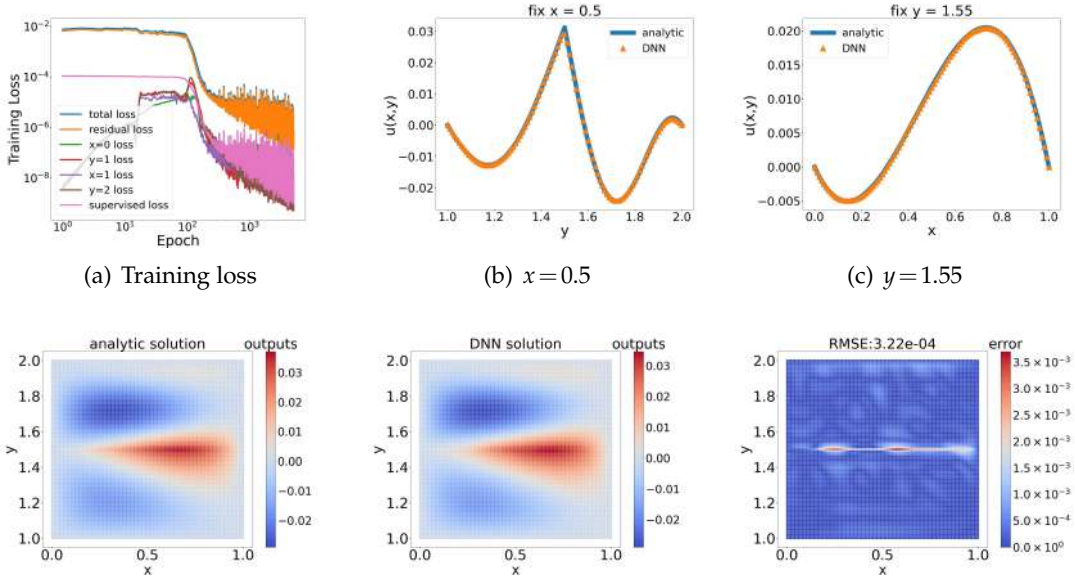
(d) Analytic solution (e) MOD-Net solution (f) Difference between two solutions

Figure 10: Example 3. Trained by only the 10×10 labeled data. (a) Training loss. The blue curve is the supervised risk Eq. (5.4). (b,c) Comparison between analytic solution and MOD-Net solution on $x=0.5$ and $y=1.55$. (d,e,f) Comparison between analytic solution and MOD-Net solution on 101×101 grid points.

Fig. 9(a). For visualizing the performance of our well-trained MOD-Net u_θ intuitively, we plot the MOD-Net solution and the corresponding analytic solution on $x=0.5$ and $y=1.55$. As shown in Fig. 9(b,c), the MOD-Net solution is not matching with the analytic solution. We also show the analytic solution and MOD-Net solution on 101×101 equidistributed isometric grid points by color. To compare these two solutions more intuitively, we calculate the difference between the two solutions at each point and show the error by color, i.e., as the error increases, the color changes from blue to red, in Fig. 9(d,e,f). The root mean square error (RMSE) is $\sim6.90\times10^{-3}$ and relative error is $\sim57.61\%$.

Then, we set the λ_1, λ_2 be zero and train the MOD-Net only by the 10×10 equidistributed isometric labeled data, i.e., 10 equidistributed isometric points in x direction and 10 equidistributed isometric points in y direction. The empirical risk, i.e., training loss is shown in Fig. 10(a). Similarly, to visualize the performance of the obtained solution u_θ, we plot the MOD-Net solution and the corresponding analytic solution on $x=0.5$ and $y=1.55$. As shown in Fig. 10(b,c), the MOD-Net solution deviates from the analytic solution. We also show the analytic solution and MOD-Net solution on 101×101 equidistributed isometric grid points by color in Fig. 10(d,e,f). The root mean square error (RMSE) is $\sim1.01\times10^{-2}$ and relative error is $\sim84.24\%$.

To sum up, with the information of only PDE or only a few labeled data, the MOD-Net cannot be trained well, however, combining these two information, we can train



(a) Training loss (b) $x=0.5$ (c) $y=1.55$

(d) Analytic solution (e) MOD-Net solution (f) Difference between two solutions

Figure 11: Example 3. Trained by the governing PDE, boundary condition,and coarse grid data together. (a) Training loss. The blue curve is the total risk Eq. (5.4). The other six curves represent six terms in the total risk Eq. (5.4), respectively. (b,c) Comparison between analytic solution and MOD-Net solution on $x=0.5$ and $y=1.55$. (d,e,f) Comparison between analytic solution and MOD-Net solution on 101×101 grid points.

the MOD-Net very well. See results in Fig. 11. The root mean square error (RMSE) is $\sim 3.22\times10^{-4}$ and relative error is $\sim 2.68\%$.

6 Numerical experiments: One-dimensional radiative transfer equation

In this section, we would apply MOD-Net to solve one-dimensional steady radiative transfer equation (RTE). Consider the density of particles in a bounded domain that interact with a background through absorption and scattering processes. The density function $u(\boldsymbol{x},\boldsymbol{v})$ follows the RTE

$$\begin{aligned} &\boldsymbol{v}\cdot\nabla u(\boldsymbol{x},\boldsymbol{v})+\frac{\sigma_T(\boldsymbol{x})}{\varepsilon(\boldsymbol{x})}u(\boldsymbol{x},\boldsymbol{v})=\frac{1}{|S|}\left(\frac{\sigma_T(\boldsymbol{x})}{\varepsilon(\boldsymbol{x})}-\varepsilon(\boldsymbol{x})\sigma_a(\boldsymbol{x})\right)\int_S u(\boldsymbol{x},\boldsymbol{\xi})\mathrm{d}\boldsymbol{\xi},\\ &u(\boldsymbol{x},\boldsymbol{v})=\phi(\boldsymbol{x},\boldsymbol{v}),\quad \boldsymbol{x}\in\Gamma=\partial\Omega,\quad \boldsymbol{v}\in S,\quad \boldsymbol{v}\cdot\boldsymbol{n}_{\boldsymbol{x}}<0, \end{aligned} \tag{6.1}$$

where $\boldsymbol{x}\in\Omega\subset\mathbb{R}^d$ is the d-dimensional space variable, $\boldsymbol{v}$ is the angular variable on unit ball $S^{d-1}\subset\mathbb{R}^d$, $\boldsymbol{n}_{\boldsymbol{x}}$ is the outward normal vector at $\boldsymbol{x}$ on the boundary, σ_a is the absorption coefficient, σ_T is the total scattering coefficient and ε is Knudsen number. For simplifying

the PDE and well describing the real situation, we use the isotropic hypothesis, then σ_a and σ_T are only related to the space variable $\boldsymbol{x}$.

High-accuracy numerical methods are developed to solve RTE, e.g., Tailored Finite Point Method (TFPM). However, these numerical methods are usually based on discrete grids. To improve the accuracy, the mesh grid has to be finer and the solving time exponentially increases. Another disadvantage is that these numerical methods usually only solve one particular PDE, i.e., if we change anyone of σ_T, σ_a, ε or ϕ, we need to reuse this algorithm to solve it. To alleviate these problems, MOD-Net approach learns the operator $\mathcal{G}:(\phi,\sigma_T,\sigma_a,\varepsilon)\mapsto u$.

We consider one-dimensional RTEs. Set $\Omega=[x_L,x_R]$, $S=[-1,1]$, normal vector $n_{x_L}=-1$, $n_{x_R}=1$, we obtain

$$\begin{aligned}
&v\partial_x u(x,v)+\frac{\sigma_T(x)}{\varepsilon(x)}u(x,v)=\left(\frac{\sigma_T(x)}{\varepsilon(x)}-\varepsilon(x)\sigma_a(x)\right)\frac{1}{2}\int_{-1}^{1}u(x,\xi)\,\mathrm{d}\xi,\\
&u(x_L,v)=\phi_L(v),\quad v>0,\\
&u(x_R,v)=\phi_R(v),\quad v<0.
\end{aligned}\tag{6.2}$$

For convenience, in this paper, we fixed the $\sigma=(\sigma_T,\sigma_a,\varepsilon)$, and our goal is only to learn a operator mapping from ϕ_L,ϕ_R to the solution u of the RTE.

6.1 Use DNN to fit Green's function

Denote

$$\mathcal{L}[u]=\boldsymbol{v}\cdot\nabla u(\boldsymbol{x},\boldsymbol{v})+\frac{\sigma_T(\boldsymbol{x})}{\varepsilon(\boldsymbol{x})}u(\boldsymbol{x},\boldsymbol{v})-\frac{1}{|S|}\left(\frac{\sigma_T(\boldsymbol{x})}{\varepsilon(\boldsymbol{x})}-\varepsilon(\boldsymbol{x})\sigma_a(\boldsymbol{x})\right)\int_S u(\boldsymbol{x},\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi},$$

the RTE (6.1) can be rewritten as the following linear PDE,

$$\begin{cases}
\mathcal{L}[u](\boldsymbol{x},\boldsymbol{v})=0, & \boldsymbol{x}\in\Omega,\quad \boldsymbol{v}\in S,\\
u(\boldsymbol{x},\boldsymbol{v})=\phi(\boldsymbol{x},\boldsymbol{v}), & \boldsymbol{x}\in\partial\Omega,\quad \boldsymbol{v}\cdot\boldsymbol{n}_{\boldsymbol{x}}<0,
\end{cases}\tag{6.3}$$

which is a special case of linear PDE (3.1).

With Green's function method, the solution of (6.3), can be represented by the following formula,

$$u(\boldsymbol{x},\boldsymbol{v};\phi,\sigma)=\int_{\partial\Omega}\int_{S\cap\{\boldsymbol{v}'|\boldsymbol{v}'\cdot\boldsymbol{n}_{\boldsymbol{x}'}<0\}}G(\boldsymbol{x},\boldsymbol{x}',\boldsymbol{v},\boldsymbol{v}')\phi(\boldsymbol{x},\boldsymbol{v})\,\mathrm{d}\boldsymbol{v}'\,\mathrm{d}\boldsymbol{x}',\tag{6.4}$$

where $G(\boldsymbol{x},\boldsymbol{x}',\boldsymbol{v},\boldsymbol{v}')$ is the solution of the following PDE,

$$\begin{cases}
\mathcal{L}[G](\boldsymbol{x},\boldsymbol{v})=0, & \boldsymbol{x}\in\Omega,\quad \boldsymbol{v}\in S,\\
G(\boldsymbol{x},\boldsymbol{x}',\boldsymbol{v},\boldsymbol{v}')=\delta(\boldsymbol{v}-\boldsymbol{v}')\delta(\boldsymbol{x}-\boldsymbol{x}'), & \boldsymbol{x},\boldsymbol{x}'\in\partial\Omega,\quad \boldsymbol{v}\cdot\boldsymbol{n}_{\boldsymbol{x}}<0,\quad \boldsymbol{v}'\cdot\boldsymbol{n}_{\boldsymbol{x}'}<0.
\end{cases}$$

Since in the one-dimensional example (6.2), we consider $\Omega=[x_L,x_R]$, then x' in $G(x,x',v,v')$ only have two values. We can rewrite the integral in (6.4) by the following formula,

$$u(x,v;\phi_L,\phi_R)=\int_0^1 G_L(x,v,v')\phi_L(v')\,\mathrm{d}v'+\int_{-1}^0 G_R(x,v,v')\phi_R(v')\,\mathrm{d}v', \tag{6.5}$$

where G_L, G_R are the solution of the following two PDEs respectively,

$$\begin{cases} v\partial_x G_L+\dfrac{\sigma_T}{\varepsilon}G_L=\left(\dfrac{\sigma_T}{\varepsilon}-\varepsilon\sigma_a\right)\dfrac{1}{2}\displaystyle\int_{-1}^1 G_L(x,\xi,v')\,\mathrm{d}\xi, & \\ G_L(x_L,v,v')=\delta(v-v'), & v>0, \\ G_L(x_R,v,v')=0, & v<0, \end{cases}$$

$$\begin{cases} v\partial_x G_R+\dfrac{\sigma_T}{\varepsilon}G_R=\left(\dfrac{\sigma_T}{\varepsilon}-\varepsilon\sigma_a\right)\dfrac{1}{2}\displaystyle\int_{-1}^1 G_R(x,\xi,v';\sigma)\,\mathrm{d}\xi, & \\ G_R(x_L,v,v';\sigma)=0, & v>0, \\ G_R(x_R,v,v';\sigma)=\delta(v-v'), & v<0. \end{cases}$$

With the help of Green's function method, to achieve our goal of fitting the operator $\mathcal{G}:(\phi_L,\phi_R)\mapsto u$, where u is the solution of (6.2) with given ϕ_L,ϕ_R, We fit $\mathcal{G}_L,\mathcal{G}_R$ with DNNs of hidden layer size 128-256-256-128 equipped with activation function tanh, i.e., a DNN $G_{\boldsymbol{\theta}_L}(x,v,v')$ is trained to represent $G_L(x,v,v')$, similarly, another DNN $G_{\boldsymbol{\theta}_R}(x,v,v')$ is for $G_R(x,v,v')$. Since the solution reflects the distribution function, we use the exponential function to make sure that G_L and G_R are positive.

In Eq. (6.5), when we calculate the 1D integral, similarly we use the Gauss-Legendre quadrature. Then we obtain the following representation of MOD-Net solution,

$$\begin{aligned} u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L,\phi_R)=&\sum_{v'\in S_{G_v}^+}\omega_{v'_+}G_{\boldsymbol{\theta}_L}(x,v,v'_+)\phi_L(v'_+) \\ &+\sum_{v'\in S_{G_v}^-}\omega_{v'_-}G_{\boldsymbol{\theta}_R}(x,v,v'_-)\phi_R(v'_-), \end{aligned} \tag{6.6}$$

where $S_{G_v}^+\subset[0,1]$, $S_{G_v}^-\subset[-1,0]$ consist of fixed points determined by Gauss-Legendre quadrature and $\omega_{v'_+}$, $\omega_{v'_-}$ are corresponding coefficients.

6.2 Empirical risk function

For one-dimensional case, to train the neural networks, we would utilize the information of PDE, i.e., governing equation and boundary condition, and a few data $S^{u,k}=\{x_i,v_i,u^k(x_i,v_i)\}_{i\in[n_k]}$ for each $\{\phi_L^k,\phi_R^k\}$, $k=1,2,\cdots,K$, where $u^k(\cdot)=u(\cdot;\phi_L^k,\phi_R^k)$. Note that $S^{u,k}$ can be numerically solved by TFPM on coarse grid points, which is not computationally expensive. For each k, we uniformly sample a set of (x,v) from $\Omega\times S=[x_L,x_R]\times[-1,1]$, i.e., $S^{\Omega,S,k}$ and uniformly sample a set of data from boundaries $\partial\Omega_L=\{(x_L,v)\}_{v\in[0,1]}$, $\partial\Omega_R=\{(x_R,v)\}_{v\in[-1,0]}$, respectively, i.e., $S^{\partial\Omega_L,k}$,$S^{\partial\Omega_R,k}$.

We use the general definition of empirical risk (3.6) for one-dimensional RTE. The empirical risk of solving RTE is as follows,

$$\begin{aligned}R_S=\frac{1}{K}\sum_{k\in[K]}\Big(&\lambda_1\frac{1}{|S^{\Omega,S,k}|}\sum_{(x,v)\in S^{\Omega,S,k}}\|v\partial_x u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)+\frac{\sigma_T(x)}{\varepsilon(x)}u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)\\&-\left(\frac{\sigma_T(x)}{\varepsilon}-\varepsilon\sigma_a(x)\right)\frac{1}{2}\int_{-1}^{1}u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,\xi;\phi_L^k,\phi_R^k)\,\mathrm{d}\xi\|_2^2\\&+\lambda_{21}\frac{1}{|S^{\partial\Omega_L,k}|}\sum_{(x,v)\in S^{\partial\Omega_L,k}}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)-\phi_L^k(v)\|_2^2\\&+\lambda_{22}\frac{1}{|S^{\partial\Omega,k}|}\sum_{(x,v)\in S^{\partial\Omega_R,k}}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)-\phi_R^k(v)\|_2^2\\&+\lambda_3\frac{1}{n_k}\sum_{i\in[n_k]}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x_i,v_i;\phi_L^k,\phi_R^k)-u^k(x_i,v_i)\|_2^2\Big).\end{aligned}$$

For integral term in the first risk term related to the governing equation of PDE, we use the Gauss-Legendre numerical integral method. We obtain

$$\begin{aligned}R_S=\frac{1}{K}\sum_{k\in[K]}\Big(&\lambda_1\frac{1}{|S^{\Omega,S,k}|}\sum_{(x,v)\in S^{\Omega,S,k}}\|v\partial_x u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)+\frac{\sigma_T(x)}{\varepsilon(x)}u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)\\&-\left(\frac{\sigma_T(x)}{\varepsilon}-\varepsilon\sigma_a(x)\right)\frac{1}{2}\sum_{\xi\in S_v}\omega_\xi u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,\xi;\phi_L^k,\phi_R^k)\|_2^2\\&+\lambda_{21}\frac{1}{|S^{\partial\Omega_L,k}|}\sum_{(x,v)\in S^{\partial\Omega_L,k}}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)-\phi_L^k(v)\|_2^2\\&+\lambda_{22}\frac{1}{|S^{\partial\Omega,k}|}\sum_{(x,v)\in S^{\partial\Omega_R,k}}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)-\phi_R^k(v)\|_2^2\\&+\lambda_3\frac{1}{n_k}\sum_{i\in[n_k]}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x_i,v_i;\phi_L^k,\phi_R^k)-u^k(x_i,v_i)\|_2^2\Big),\end{aligned}\tag{6.7}$$

where $S_v\subset[-1,1]$ consists of fixed integration points, determined by Gauss-Legendre quadrature and ω_ξ's are corresponding coefficients.

In practical applications, it is usually not easy to measure $u(x,v)$, but density $\rho(x)=\frac{1}{2}\int_{-1}^{1}u(x,\xi)\mathrm{d}\xi$ can be measured. Therefore, we would utilize the information of PDE and a few data $S^{\rho,k}=\{x_i,\ \rho^k(x_i)\}_{i\in[n_k]}$ for each $\{\phi_L^k,\phi_R^k\}$, $k=1,2,\cdots,K$, where $\rho^k(\cdot)=$

$\frac{1}{2}\int_{-1}^{1} u\left(\cdot,\xi;\phi_L^k,\phi_R^k\right)\mathrm{d}\xi$. And we obtain another empirical risk,

$$\begin{aligned}R_S=\frac{1}{K}\sum_{k\in[K]}\Big(&\lambda_1\frac{1}{|S^{\Omega,S,k}|}\sum_{(x,v)\in S^{\Omega,S,k}}\|v\partial_x u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)+\frac{\sigma_T(x)}{\varepsilon(x)}u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)\\&-\left(\frac{\sigma_T(x)}{\varepsilon}-\varepsilon\sigma_a(x)\right)\frac{1}{2}\sum_{\xi\in S_v}\omega_\xi u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,\xi;\phi_L^k,\phi_R^k)\|_2^2\\&+\lambda_{21}\frac{1}{|S^{\partial\Omega_L,k}|}\sum_{(x,v)\in S^{\partial\Omega_L,k}}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)-\phi_L^k(v)\|_2^2\\&+\lambda_{22}\frac{1}{|S^{\partial\Omega,k}|}\sum_{(x,v)\in S^{\partial\Omega_R,k}}\|u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L^k,\phi_R^k)-\phi_R^k(v)\|_2^2\\&+\lambda_3\frac{1}{n_k}\sum_{i\in[n_k]}\|\frac{1}{2}\sum_{\xi'\in S_v'}\omega_{\xi'}u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x_i,\xi';\phi_L^k,\phi_R^k)-\rho^k(x_i)\|_2^2\Big),\end{aligned}\tag{6.8}$$

where $S_v,S_v'\subset[-1,1]$ consists of fixed integration points, determined by Gauss-Legendre quadrature and $\omega_\xi,\omega_{\xi'}$'s are corresponding coefficients.

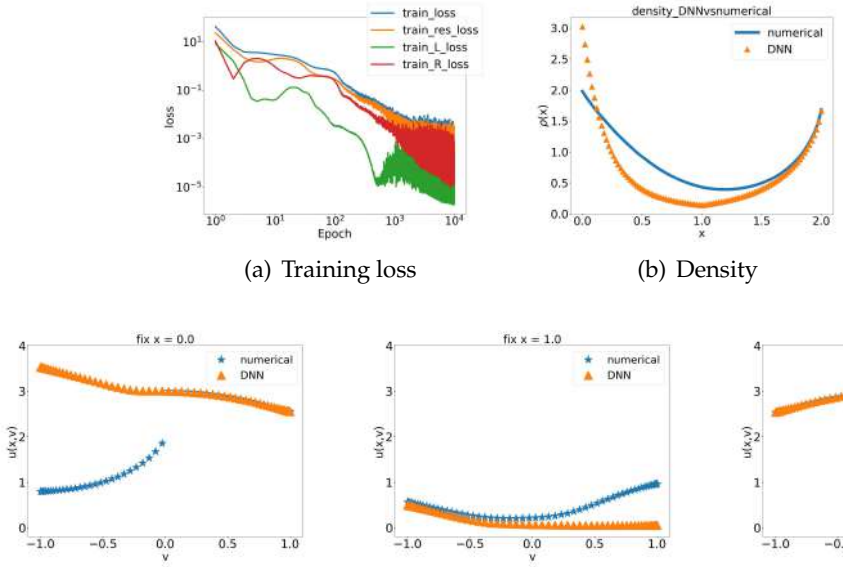
6.3 Learning process

During training, for each epoch, we first randomly choose $\{(\phi_L^k,\phi_R^k)\}_{k\in[K]}$ and calculate the values of boundary condition ϕ_L^k, ϕ_R^k on fixed integration points $v_+'\in S_{G_v}^+$, $v_-'\in S_{G_v}^-$ respectively. Second, we randomly sample points and obtain the data set $S^{\Omega,S,k}$, $S^{\partial\Omega_L,k}$, $S^{\partial\Omega_R,k}$. We obtain the data set $D_L=\{(x,v,v_+',\phi_L^k(v_+'))|(x,v)\in S^{\Omega,S,k}\cup S^{\partial\Omega_L,k}\cup S^{\partial\Omega_R,k},\ v_+'\in S_{G_v}^+\}$ and $D_R=\{(x,v,v_-',\phi_R^k(v_-'))|(x,v)\in S^{\Omega,S,k}\cup S^{\partial\Omega_L,k}\cup S^{\partial\Omega_R,k},\ v_-'\in S_{G_v}^-\}$. In the following, for each k, we feed the data D_L, D_R into the neural network $G_{\boldsymbol{\theta}_L}(x,v,v')$, $G_{\boldsymbol{\theta}_R}(x,v,v')$ respectively, and calculate the empirical risk (6.7) or (6.8) defined utilizing the governing equation of PDE, boundary condition and a few labeled data $S^{u,k}$. We train neural network $G_{\boldsymbol{\theta}_L}$ and $G_{\boldsymbol{\theta}_R}$ with Adam to minimize the empirical risk, and finally obtain well-trained Green's function DNNs $G_{\boldsymbol{\theta}_L}$ and $G_{\boldsymbol{\theta}_R}$, furthermore, according to (6.6), we obtain a neural operator $u_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2}(x,v;\phi_L,\phi_R)$.

6.4 Results

To see the performance of obtained operator, we need the reference solution. Since it is difficult to obtain the analytical solution of RTE, we use the TFPM method to obtain the numerical solution as reference.

Example 4. MOD-Net with data-regularization for radiative transfer equation with σ varying with x. Only for illustration of data regularization in solving RTE, we consider



(a) Training loss (b) Density

(c) Solution on $x=0$ (d) Solution on $x=1$ (e) Solution on $x=2$

Figure 12: Example 4. Trained by only the PDE,i.e., the governing equation and boundary condition. (a) Training loss. The blue curve is the total risk Eq. (6.7). The other three curves represent three terms in the total risk Eq. (6.7) except the last term, respectively. (b) Comparison between numerical solution and MOD-Net solution on density. (c,d,e) Comparison between numerical solution and MOD-Net solution on $x=0,1,2$.

a simple case, $x_L=0$, $x_R=2$, and

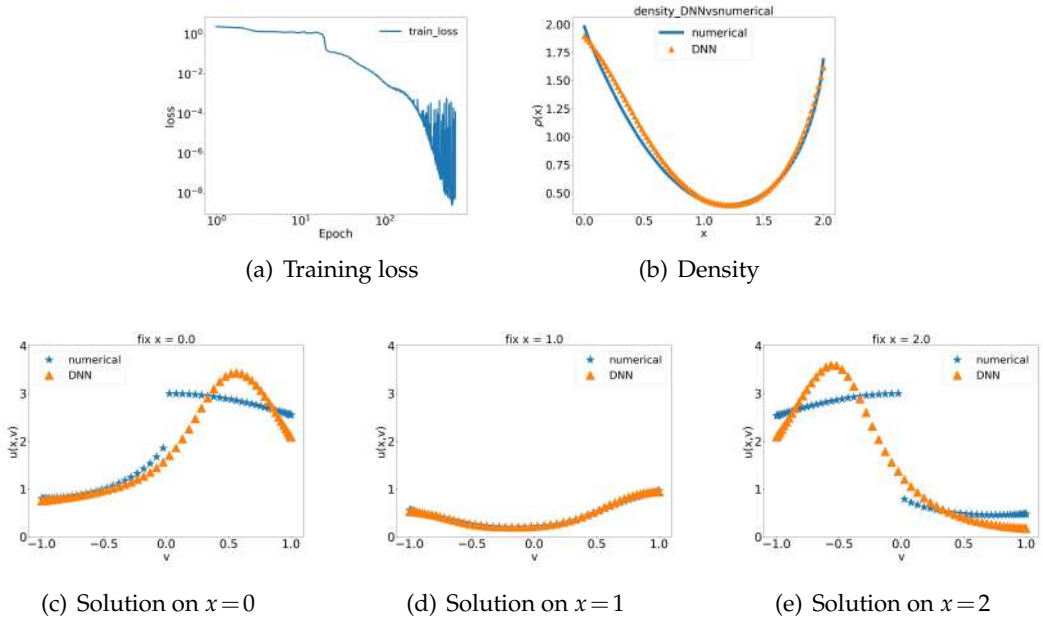
$$\sigma_T(x)=\begin{cases} x+1, & 0\le x<1, \\ 2, & 1\le x\le 2, \end{cases} \tag{6.9}$$

and

$$\sigma_a(x)=\begin{cases} x, & 0\le x<1, \\ 1, & 1\le x\le 2, \end{cases} \tag{6.10}$$

the graphs of which are shown in Fig. 16(a). To avoid the multi-scale phenomenon, we take $\varepsilon(x)=1$. We set the boundary condition $\phi_L=\phi_R=a_1\cos(\omega v)+a_2\sin(\omega v)+2$, which is determined by a_1,a_2,ω. In fact, many boundary conditions can be represented with these basic functions. For convenience, we set $\omega=1,a_1=1$ and only change a_2. Then the boundary condition $\phi_L=\phi_R=\cos(v)+a_2\sin(v)+2$.

Fix $a_2=0.01$, by TFPM method, we obtain 20 labeled data $S^{u,k}=\{x_i,v_i,u^k(x_i,v_i)\}_{i=1}^{20}$ on coarse grids, i.e., 5 equidistributed isometric points in x direction and 4 equidistributed isometric points in v direction and 10 labeled data $S^{\rho,k}=\{x_i,\rho^k(x_i)\}_{i=1}^{10}$, where $\rho^k(x_i)=\frac{1}{2}\sum_{\xi'\in S'_v}\omega_{\xi'}u^k(x_i,\xi')$ and S'_v consists of fixed Gauss-Legendre integration points, $|S'_v|=30$, and $\omega_{\xi'}$'s are corresponding coefficients. For this example, we train the solver of the RTE with four different loss functions.



(a) Training loss (b) Density

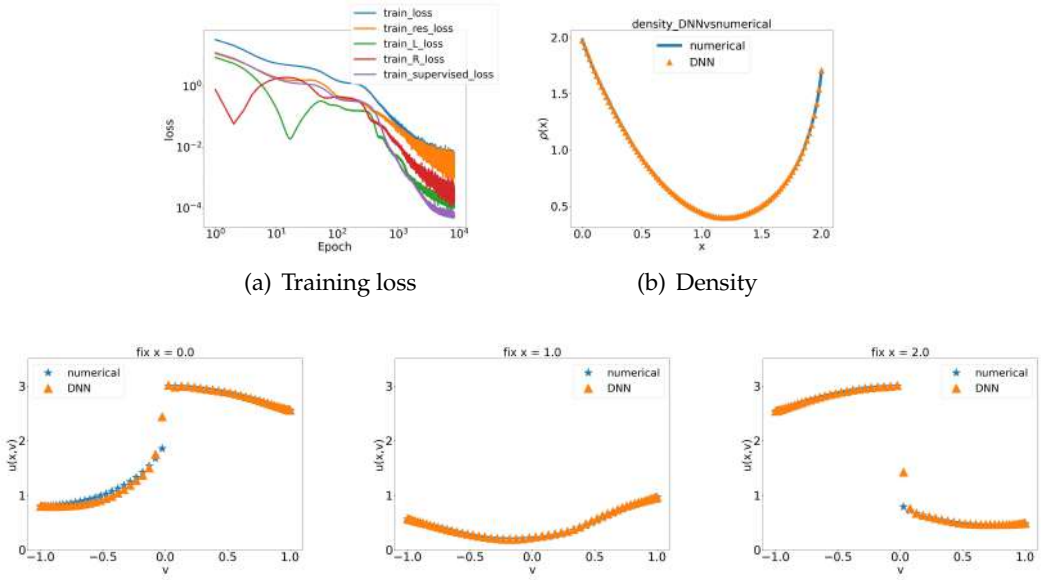
(c) Solution on $x=0$ (d) Solution on $x=1$ (e) Solution on $x=2$

Figure 13: Example 4. Trained by only the labeled data. (a) Training loss. The blue curve is the last term in total risk Eq. (6.7). The other three curves represent three terms in the total risk Eq. (6.7) except the last term, respectively. (b) Comparison between numerical solution and MOD-Net solution on density. (c,d,e) Comparison between numerical solution and MOD-Net solution on $x=0,1,2$.

First, we set the λ_3 in (6.7) be zero, i.e., training the MOD-Net only by the governing equation and boundary conditions, the empirical risk, i.e., training loss is shown in Fig. 12(a). To test the performance of the obtained MOD-Net solution $u_{\boldsymbol{\theta}}$, we calculate the density $\rho(x)$ of u. The density of the MOD-Net solution and numerical TFPM solution with equidistributed isometric grids $(x_i,v_j)_{i\in[101],j\in[60]}$ are significantly different, as shown in Fig. 12(b). For visualization, we plot the MOD-Net solution and the corresponding numerical solution on $x=0,1,2$. As shown in Fig. 12(c, d, e), the MOD-Net solution is far from the numerical solution.

Second, $\lambda_1,\lambda_{21},\lambda_{22}$ in (6.7) are set as zero, i.e., training the MOD-Net by only the 20 labeled data with the training loss shown in Fig. 13(a). The training loss (indicated by blue curve) decays with training epoch except oscillation appears in the final stage. We use the trick of early stopping with tolerance 50 to obtain a DNN with small loss. Similarly, to test the performance of the obtained MOD-Net solution $u_{\boldsymbol{\theta}}$, we calculate the density $\rho(x)$ of u. The density of the MOD-Net solution and TFPM solution have clear difference as shown in Fig. 13(b). For visualization, we plot the MOD-Net solution and the corresponding numerical solution on $x=0$. As shown in Fig. 13(c), the MOD-Net solution significantly deviates from the numerical solution.

Third, we train the MOD-Net by the governing equation, boundary condition and 20 labeled data $S^{u,k}$ simultaneously, and $\lambda_1,\lambda_{21},\lambda_{22},\lambda_3$ in (6.7) are set as one. The training



(a) Training loss (b) Density

(c) Solution on $x=0$ (d) Solution on $x=1$ (e) Solution on $x=2$

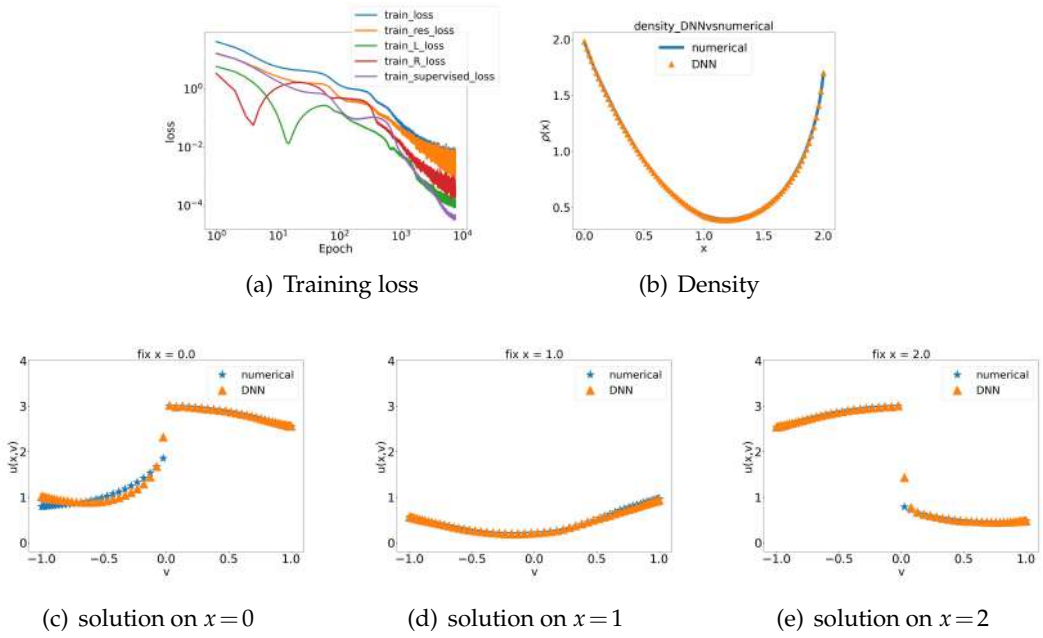
Figure 14: Example 4. Trained by the governing PDE, boundary condition, and coarse grid data together. (a) Training loss. The blue curve is the total risk Eq. (6.7). The other four curves represent four terms in the total risk Eq. (6.7), respectively. (b) Comparison between numerical solution and MOD-Net solution on density. (c,d,e) Comparison between numerical solution and MOD-Net solution on $x=0,1,2$.

total loss (6.7) decays with training epoch as shown in Fig. 14(b). Similarly, we calculate the density of MOD-Net solution and TFPM solution. They are very consistent as shown in Fig. 14(c). As shown in Fig. 14(d,e,f), the MOD-Net solution overlap with the numerical solution very well except for the discontinuous points.

Four, we train the MOD-Net by the governing equation, boundary condition and 10 labeled data $S^{\rho,k}$ simultaneously, and $\lambda_1,\lambda_{21},\lambda_{22},\lambda_3$ in (6.8) are set as one. The training total loss (6.8) decays with training epoch as shown in Fig. 15(b). To see the performance of the trained MOD-Net, we calculate the density of MOD-Net solution and TFPM solution. They are very consistent as shown in Fig. 15(c). As shown in Fig. 15(d,e,f), the MOD-Net solution overlap with the numerical solution well except for the discontinuous points.

To sum up, with the information of only PDE or only a few labeled data, the MOD-Net cannot be trained well, however, combining these two information, we can train the MOD-Net very well. Note that the labeled data is not restricted in the solution, the information of the density can also benefit the training of MOD-Net.

To solve the PDE with various a_2 accurately and quickly, we use MOD-Net approach to learn a neural operator. For this case, we train MOD-Net with PDE and a few labeled data simultaneously. For each a_2 in $\{0.03k\}_{k=1}^{33}$, by TFPM method, we calculate the corresponding numerical solution on 20 fixed coarse grids, i.e., 5 equidistributed isometric



(a) Training loss (b) Density

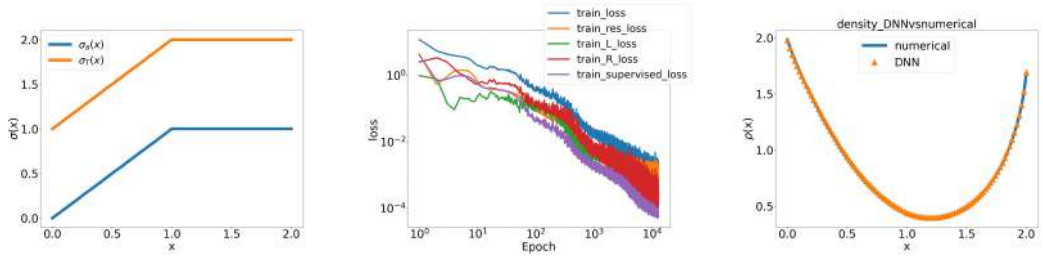
(c) solution on $x=0$ (d) solution on $x=1$ (e) solution on $x=2$

Figure 15: Example 4. Trained by the governing PDE, boundary condition, and coarse density ρ together. (a) Training loss. The blue curve is the total risk Eq. (6.8). The other four curves represent four terms in the total risk Eq. (6.8), respectively. (b) Comparison between numerical solution and MOD-Net solution on density. (c,d,e) Comparison between numerical solution and MOD-Net solution on $x=0,1,2$.

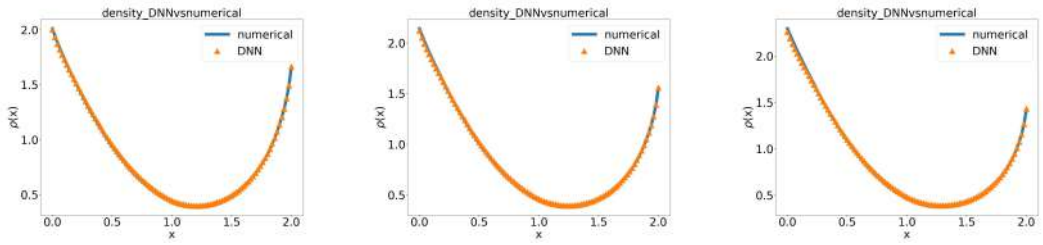
points in x direction and 4 equidistributed isometric points in v direction, as labeled data and obtain $S^{u,k}=\{x_i,v_i,u^k(x_i,v_i)\}_{i=1}^{20}$.

We set K=20 and for each epoch, we sample K different a_2's uniformly from $\{0.03k\}_{k=1}^{33}$. In training, we set the number of integration points $|S_{G_v}^+|=30$, $|S_{G_v}^-|=30$, $|S_v|=30$, and for each epoch, for each k, we uniformly sample 500 points in $[0,2]\times[-1,1]$ for $S^{\Omega,S,k}$, 500 points v_i's in $[0,1]$ for $S^{\partial\Omega_L,k}=\{(0,v_i)|i=1,\cdots,500\}$, another 500 points v_i's in $[-1,0]$ for $S^{\partial\Omega_R,k}=\{(2,v_i)|i=1,\cdots,500\}$ and use the 20 labeled data in $S^{u,k}$. The training loss is shown in Fig. 16(b). The total risk (indicated by blue curve) decays with training epoch. For visualization of each part of the total risk, we also display the residual loss of the RTE (indicated by orange curve), the loss of two boundary lines (indicated by green and red curves) and the loss of supervised data (indicated by remaining curve), which decay with the training epoch overall.

To test the performance of the trained MOD-Net on $a_2=0.02,0.11,0.51,0.99$. For each a_2, we calculate the density $\rho(x)$ of the solution. For all a_2's, the density of MOD-Net solution and numerical solution by TFPM method with equidistributed isometric grids $(x_i,v_j)_{i\in[101],j\in[60]}$ are very consistent as shown in Fig. 16.



(a) $\sigma(x)$ (b) Training loss (c) $a_2=0.02$



(d) $a_2=0.11$ (e) $a_2=0.51$ (f) $a_2=0.99$

Figure 16: Example 4. Training MOD-Net by the governing PDE, boundary condition, and coarse grid data together. (a) $\sigma_T(x)$ and $\sigma_a(x)$ vs. x. (b) Training loss vs. epoch for the learning of the operator of RTE with continuous $\sigma_T(x)$, $\sigma_a(x)$. The blue curve is the total risk Eq. (6.7). The other four curves represent four terms in the total risk Eq. (6.7), respectively. (c,d,e,f) Comparison between numerical solution and MOD-Net solution on density corresponding to each of four boundary conditions $\phi_L=\phi_R=\cos(v)+a_2\sin(v)+2$ determined by $a_2=0.02,0.11,0.51,0.99$.

7 Numerical experiments: 1D Burgers equation

To show the performance of the MOD-Net for a nonlinear PDE, first we consider the 1D Burgers equation in the steady state,

$$\begin{aligned}&\partial_x\left(\frac{1}{2}u^2(x)\right)=\nu\partial_{xx}u(x)+g(x),\quad x\in\Omega,\\&u(x)=\phi(x),\quad x\in\partial\Omega,\end{aligned}\tag{7.1}$$

For illustration, we take $\Omega=[-1,1]$, $\nu=1$,$g(x)=0$ and $\phi(x)=c_1\cos(k_1x)+c_2\sin(k_2x)$ for all $x\in\Omega$,

$$\begin{aligned}&\partial_x\left(\frac{1}{2}u^2(x)\right)=\partial_{xx}u(x),\quad x\in\Omega,\\&u(-1)=c_1\cos(-k_1)+c_2\sin(-k_2),\\&u(1)=c_1\cos(k_1)+c_2\sin(k_2),\end{aligned}\tag{7.2}$$

in which c_1,c_2,k_1,k_2 control the boundary condition. For this problem, it is difficult to obtain the analytical solution, but a lot of traditional numerical schemes can be used to solve it. In this paper, we use the upwind scheme.

7.1 Use DNN to fit nonlinear operator

For a linear PDE, Green's function can help us obtain the solution of PDE due to the superposition principle. For a nonlinear PDE, similarly we use the following representation of the solution of PDE (7.1),

$$u(x;\phi,g)=F\left(\int_\Omega G_1(x,x')g(x')\,\mathrm{d}x'+\int_{\partial\Omega}G_2(x,x')\phi(x')\,\mathrm{d}x'\right). \tag{7.3}$$

In the considered example, $\Omega=[-1,1]$ and $g(x)=0$, we have

$$\begin{aligned}u(x;\phi)&=F\left(\int_{\partial\Omega}G_2(x,x')\phi(x')\,\mathrm{d}x'\right)\\&=F(G_2(x,-1)\phi(-1)+G_2(x,1)\phi(1)).\end{aligned} \tag{7.4}$$

We use a DNN of hidden layer size 256-256-256-256 equipped with sigmoid function as activation function $G_{\theta_L}(x)$ to fit $G_2(x,-1)$, a DNN with the same setting $G_{\theta_R}(x)$ to fit $G_2(x,1)$ and a DNN of one hidden layer, the size of which is 256, equipped with sigmoid function as activation function $F_\theta(x)$. Then we can represent neural operator $u_\theta(x;g)$ with DNN $G_{\theta_L}(x)$, $G_{\theta_R}(x)$ and $F_\theta(x)$, that is,

$$u_\theta(x;\phi)=F_\theta(G_{\theta_L}(x)\phi(-1)+G_{\theta_R}(x)\phi(1)). \tag{7.5}$$

7.2 Empirical risk function

For this simple example, to learn a neural operator, which can approximate the operator $\mathcal{G}:\phi\mapsto u$, we use no labeled data and only utilize the information of PDE, i.e., governing equation and boundary condition, ϕ^k, $k=1,2,\cdots,K$.

To utilize the constraint of governing equation of PDE, we uniformly sample a set of data from $\Omega=[-1,1]$, i.e., $S^{\Omega,k}$. Since the boundary $\partial\Omega$ consists of two points, i.e. -1 and 1, then we directly use the information of these two points.

The empirical risk for this example is as follows,

$$\begin{aligned}R_S=\frac{1}{K}\sum_{k\in[K]}\Bigg(&\lambda_1\frac{1}{|S^{\Omega,k}|}\sum_{x\in S^{\Omega,k}}\Big(\partial_x(\frac{1}{2}u_\theta^2(x;\phi^k))-\partial_{xx}u_\theta(x;\phi^k)\Big)^2\\&+\lambda_2\Big(u_\theta(-1;\phi^k)-\phi^k(-1))\Big)\\&+\lambda_3\Big(u_\theta(1;\phi^k)-\phi^k(1))\Big)\Bigg).\end{aligned} \tag{7.6}$$

7.3 Learning process

For each training epoch, we first randomly choose source functions denoted by $\{\phi^k\}_{k=1}^K$. Second, we randomly sample data and obtain data set $S^{\Omega,k}$. In the following, we feed the

data into the neural networks $G_{\theta_L}(x)$, $G_{\theta_R}(x)$ and calculate the total risk (7.6) with neural operator $u_\theta(x;g)$, see Eq. (7.5). Train neural networks $G_{\theta_L}(x)$, $G_{\theta_R}(x)$, and $F_\theta(x)$ with Adam to minimize the total risk, and finally we obtain well-trained DNNs $G_{\theta_L}(x)$, $G_{\theta_R}(x)$ and $F_\theta(x)$, furthermore, according to Eq. (7.5), we obtain a neural operator $u_\theta(x;\phi)$.

7.4 Results

The boundary condition $\phi(x)=c_1\cos(k_1x)+c_2\sin(k_2x)$ is determined by c_1,c_2,k_1,k_2. To illustrate our approach, we train a neural operator mapping from $\phi(x)$ to the solution $u(x)$ of the Burgers equation (7.2).

Example 5: MOD-Net for Burgers equation. To solve the PDE with various c_1,c_2,k_1,k_2 accurately and quickly, we use MOD-Net approach to learn a neural operator. For simplicity, we fixed $c_1=4$, $k_1=2$, $k_2=10$ and only change c_2.

In training process, we set $K=21$ and for each epoch, we choose c_2's, i.e., $\{3+0.2(k-1)\}_{k=1}^{21}$. For each k, we uniformly sample 500 points in $[-1,1]$ for $S^{\Omega,S,k}$. The training loss is shown in Fig. 17. The total risk (indicated by blue curve) roughly decays with training epoch. For visualization of each part of the total risk, we also display the residual loss of the Burgers equation (indicated by orange curve) and the loss of two boundary loss (indicated by green and red curves), which roughly decay with the training epoch at the final stage.

To test the performance of the trained MOD-Net on $c_2=3.1,4.1,5.1,6.1,6.9$. For each c_2, we plot the MOD-Net solution and the corresponding numerical solution on 500 equidistributed isometric points sampled in $\Omega=[-1,1]$. As shown in Fig. 18, the MOD-Net solutions overlap well with the numerical solutions for all cases.

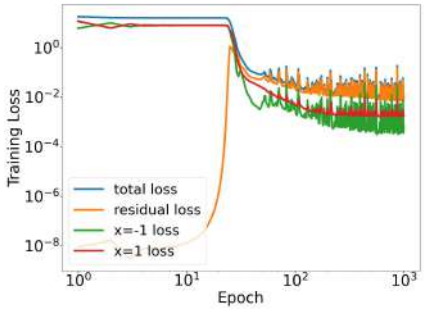
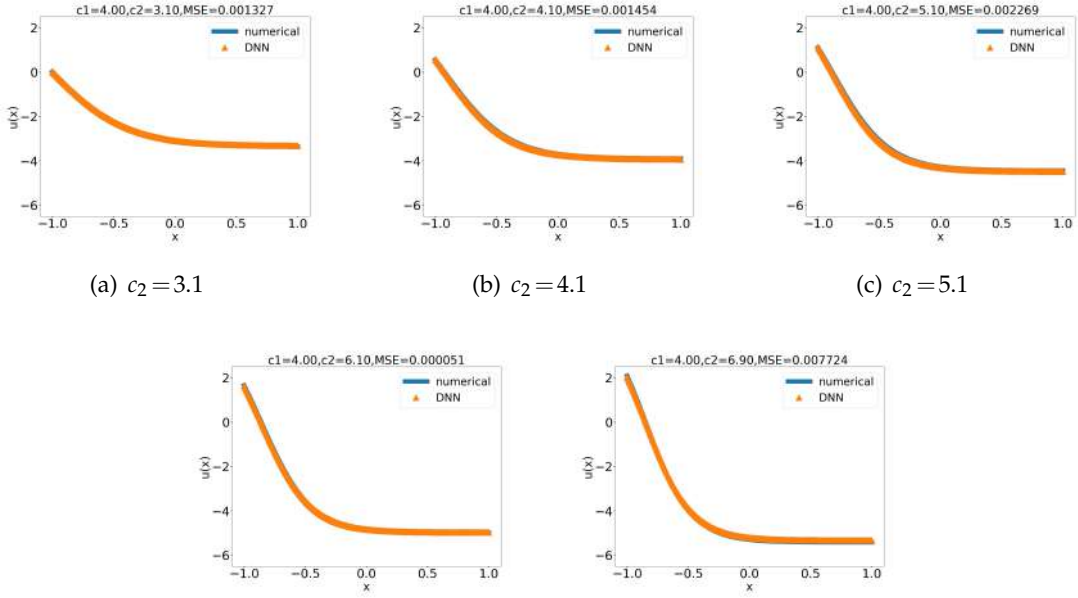


Figure 17: Example 5. Training loss vs. epoch for the learning of the operator of Burgers equation. The blue curve is the total risk Eq. (7.6). The other three curves represent three terms in the total risk Eq. (7.6), respectively.



(a) $c_2=3.1$ (b) $c_2=4.1$ (c) $c_2=5.1$

(d) $c_2=6.1$ (e) $c_2=6.9$

Figure 18: Example 5. Comparison between numerical solution and MOD-Net solution on equidistributed isometric points in $\Omega=[-1,1]$ corresponding to each of boundary conditions determined by $\phi(x)=4\cos(2x)+c_2\sin(10x)$.

8 Numerical experiments: 2D nonlinear equation

We consider the following nonlinear equation,

$$\begin{aligned} &-\Delta u(\boldsymbol{x})+0.01u^3(\boldsymbol{x})=g(\boldsymbol{x}), \quad \boldsymbol{x}\in\Omega,\\ &u(\boldsymbol{x})=0, \quad \boldsymbol{x}\in\partial\Omega. \end{aligned} \tag{8.1}$$

Take a 2D case as example, in which source function $g(x,y)=-a(x^2-x+y^2-y)+0.01(\frac{a}{2}x(x-1)y(y-1))^3$,

$$\begin{aligned} &-(\partial_{xx}u+\partial_{yy}u)+0.01u^3=-a(x^2-x+y^2-y)+0.01\Big(\frac{a}{2}x(x-1)y(y-1)\Big)^3, \quad (x,y)\in\Omega,\\ &u=0, \quad (x,y)\in\partial\Omega, \end{aligned} \tag{8.2}$$

where $\Omega=[0,1]^2$ and a controls the source term $g(x,y)$. Obviously, the analytical solution is $u(x,y;g)=\frac{a}{2}x(x-1)y(y-1)$.

8.1 Use DNN to fit nonlinear operator

In the considered 2D case, $g(x,y)=-a(x^2-x+y^2-y)+0.01(\frac{a}{2}x(x-1)y(y-1))^3$, we have

$$u(x,y;g)=F(\int_0^1\int_0^1 G(x,y,x',y')g(x',y')\mathrm{d}x'\mathrm{d}y'). \tag{8.3}$$

We use a DNN of hidden layer size 128-128-128-128 $G_{\boldsymbol{\theta}}(x,y,x',y')$ to fit the function $G(x,y,x',y')$ and a DNN of one hidden layer, the size of which is 256, equipped with sigmoid function as activation function $F_{\boldsymbol{\theta}}(x,y)$.

When we calculate the integral, We use the Gauss-Legendre numerical integral. Then we can represent neural operator $u_{\boldsymbol{\theta}}(x,y;g)$ with DNN $G_{\boldsymbol{\theta}}(x,y,x',y')$ and $F_{\boldsymbol{\theta}}(x,y)$, that is,

$$u_{\boldsymbol{\theta}}(x,y;g)=F_{\boldsymbol{\theta}}(\sum_{x'\in S_{G_x}}\sum_{y'\in S_{G_y}}\omega_{x'}\omega_{y'}G_{\boldsymbol{\theta}}(x,y,x',y')g(x',y')), \tag{8.4}$$

where $S_{G_x}\subset[0,1]$,$S_{G_y}\subset[0,1]$ consist fixed integration points, determined by 1D Gauss-Legendre quadrature and $\omega_{x'}$, $\omega_{y'}$ are corresponding coefficients.

8.2 Empirical risk function

For this example, to train the neural networks, we use no labeled data and only utilize the information of PDE, i.e., governing equation and boundary condition, for each g^k, $k=1,2,\cdots,K$. To utilize the constraint of governing equation of PDE, we uniformly sample a set of data from $\Omega=[0,1]\times[0,1]$, i.e., $S^{\Omega,k}$. To utilize the information of boundary constraint, for each k, we uniformly sample a set of data from $\partial\Omega$. Since the boundary $\partial\Omega$ consists of four line segments, i.e., $\partial\Omega=\bigcup_{i=1}^4\partial\Omega_i$, where $\partial\Omega_1=\{(0,y)\}_{y\in[0,1]}$, $\partial\Omega_2=\{(1,y)\}_{y\in[0,1]}$, $\partial\Omega_3=\{(x,0)\}_{x\in[0,1]}$, $\partial\Omega_4=\{(x,1)\}_{x\in[0,1]}$, we uniformly sample a set of data from $\partial\Omega_i$ respectively, i.e., $S^{\partial\Omega_i,k}$, $i=1,2,3,4$.

Since we use no labeled data, λ_3 in the general definition is set as zero. The empirical risk for this example is as follows,

$$\begin{aligned}R_S=\frac{1}{K}\sum_{k\in[K]}\Bigg(&\lambda_1\frac{1}{|S^{\Omega,k}|}\sum_{(x,y)\in S^{\Omega,k}}\Big(\partial_{xx}u_{\boldsymbol{\theta}}(x,y;g^k)+\partial_{yy}u_{\boldsymbol{\theta}}(x,y;g^k)+u_{\boldsymbol{\theta}}(x,y;g^k)^3-g^k(x,y)\Big)^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega_1,k}|}\sum_{(x,y)\in S^{\partial\Omega_1,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega_2,k}|}\sum_{(x,y)\in S^{\partial\Omega_2,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega_3,k}|}\sum_{(x,y)\in S^{\partial\Omega_3,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\\&+\lambda_2\frac{1}{|S^{\partial\Omega_4,k}|}\sum_{(x,y)\in S^{\partial\Omega_4,k}}u_{\boldsymbol{\theta}}(x,y;g^k)^2\Bigg).\end{aligned} \tag{8.5}$$

8.3 Learning process

For each training epoch, we first randomly choose source functions $\{g^k\}_{k=1}^K$ and calculate their values on fixed integration points(x',y'), where $x' \in S_{G_x}$ and $y' \in S_{G_y}$. Second, we randomly sample data and obtain data set $S^{\Omega,k}, S^{\partial\Omega_i,k}$, $i=1,2,3,4$. We obtain the data set $D=\{(x,y,x',y',g^k(x',y'))|(x,y) \in S^{\Omega,k} \cup \bigcup_{i=1}^4 S^{\partial\Omega_i,k}, x' \in S_{G_x}, y' \in S_{G_y}\}$. In the following, we feed the data into the neural network $G_{\boldsymbol{\theta}}(x,y,x',y')$ and calculate the total risk (8.5)) with neural operator $u_{\boldsymbol{\theta}}(x,y;g)$, see Eq. (8.4). Train neural networks $G_{\boldsymbol{\theta}}$ and $F_{\boldsymbol{\theta}}$ with Adam to minimize the total risk, and finally we obtain well-trained DNNs $G_{\boldsymbol{\theta}}$ and $F_{\boldsymbol{\theta}}$, furthermore, according to Eq. (8.4), we obtain a neural operator $u_{\boldsymbol{\theta}}(x,y;g)$.

8.4 Results

The source function $g(x,y) = -a(x^2-x+y^2-y)+0.01(\frac{a}{2}x(x-1)y(y-1))^3$ is determined by a. We then denote source function $g(x,y)$ by g_a. To illustrate our approach, we train a neural operator mapping from g to the solution of Eq. (8.2).

8.4.1 Example 6: MOD-Net for a family of nonlinear equations

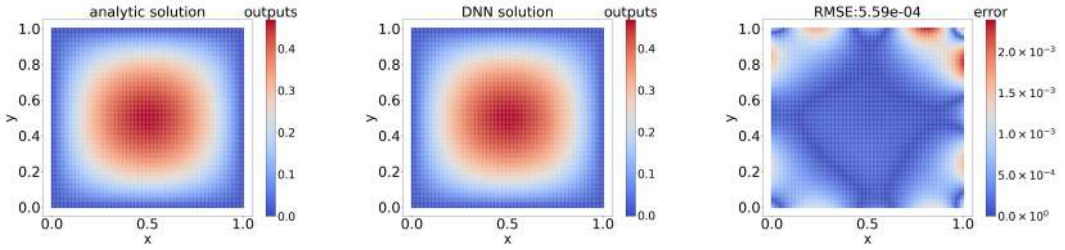
For illustration, we would train the MOD-Net by various source functions and test the MOD-Net with several source functions that are not used for training.

In training, we set $K=10$ and for each epoch, we choose a family of source functions $g_a(x,y) = -a(x^2-x+y^2-y)+0.01(\frac{a}{2}x(x-1)y(y-1))^3$, where the control parameter $a \in \{10k\}_{k=1}^{10}$. We set the number of integration points $|S_{G_x}|=10$ and $|S_{G_y}|=10$, and for each epoch, we randomly sample 200 points in $[0,1]^2$ and sample 200 points in each line of boundary, respectively. We set $\lambda_1=1, \lambda_2=1$ in the empirical risk using least square loss (8.5).

We test the performance of this well-trained MOD-Net on $a=15$. For each fixed a, the corresponding exact true solution is $u(x,y)=\frac{a}{2}x(x-1)y(y-1)$. For visualizing the performance of our well-trained MOD-Net, we show the analytic solution and MOD-Net solution on 101×101 equidistributed isometric grid points by color. To compare these two solutions more intuitively, we calculate the difference between the two solutions at each point and show the error by color, i.e., as the error increases, the color changes from blue to red, in Fig. 19. The root mean square error (RMSE) is $\sim 5.6\times10^{-4}$. We also show the solution obtained by MOD-Net and the corresponding analytical solution on fixed $x=0,0.5,1$ for considered test source function. As shown in Fig. 20, the MOD-Net can well predict the analytic solutions.

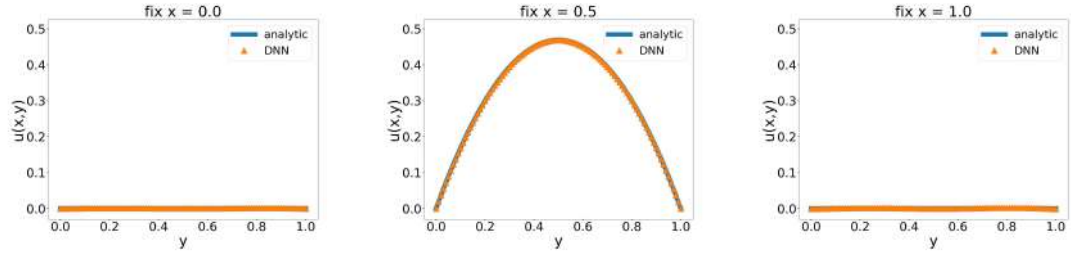
9 Conclusion and discussion

In this work, we propose a model-operator-data network (MOD-Net) for solving PDEs. The advantage of the MOD-Net is that it solves a family of PDEs but not a specific one



(a) $a=15$ (b) $a=15$ (c) $a=15$

Figure 19: Example 6. Comparison between analytic solution and MOD-Net solution on 101×101 grid points corresponding to source terms determined by $a=15$. (a) Analytic solution. (b) MOD-Net solution. (c) The difference between the analytic solution and MOD-Net solution. Since the error at most points are very small, here we reduce the upper limit of the Color bar to see more information.



(a) $a=15$, $x=0.0$ (b) $a=15$, $x=0.5$ (c) $a=15$, $x=1.0$

Figure 20: Example 6. Comparison between analytic solution and MOD-Net solution on x=0,0.5,1 corresponding to source function determined by test $a=15$.

in a meshless way without expensive labeled data. For illustration, we use MOD-Net approach solving Poisson equation and a family of nonlinear PDEs, in which the empirical risk of MOD-Net only requires the physical information of the PDE and does not require any labeled data. And in some examples, such as constructed equation, which can be regarded as a simplification of RTE, and the real one-dimensional RTE, with only physical information and operator representation built into the empirical risk is not enough to guarantee the learning of correct solutions despite the small training loss. By adding *few labeled data as regularization* which are calculated analytically or computed by traditional numerical schemes on the coarse grid points with cheap computational cost, we show that the MOD-Net can be well trained to approximate the correct solution.

Though the good results are shown above, there are many important problems of efficiently using DNN to solve PDE for future works. One is that it remains unclear what are the appropriate weights of different components to use in the total loss. Second is to understand why the learned solver can be far away from the true solution even when residual loss and boundary loss are almost zero and why data regularization can rescue this phenomenon. And Third, the MOD-Net is a framework to guide us how

to take advantage of DNN to solve the physical PDEs. The aforementioned three key components should be adapted to specific PDEs. For nonlinear operator it is still not clear what representations we should use, although many existing work propose their own representations [13, 14, 17, 22] . For the data part, how many labeled data should we use and whether noisy data will have the same good regularization effect are all very important work to be done. Other works need to be done including how we can use DNN to represent a Green's function related to variable coefficient, such as the σ in RTE. Typically traditional DNN is not possible to parameterize an operator that is imposed on functions, so a careful design and build of the architecture is needed to feed information into the network of operator. We leave all this for future discussion.

Acknowledgments

This work is sponsored by the National Key R&D Program of China Grant No. 2019YFA0709503 (Z. X.) and No. 2020YFA0712000 (Z. M.), the Shanghai Sailing Program (Z. X.), the Natural Science Foundation of Shanghai Grant No. 20ZR1429000 (Z. X.), the National Natural Science Foundation of China Grant No. 62002221 (Z. X.), the National Natural Science Foundation of China Grant No. 12101401 (T. L.), the National Natural Science Foundation of China Grant No. 12101402 (Y. Z.), Shanghai Municipal of Science and Technology Project Grant No. 20JC1419500 (Y. Z.), the Lingang Laboratory Grant No. LG-QS-202202-08 (Y. Z.), the National Natural Science Foundation of China Grant No. 12031013 (Z. M.), Shanghai Municipal of Science and Technology Major Project No. 2021SHZDZX0102, and the HPC of School of Mathematical Sciences and the Student Innovation Center at Shanghai Jiao Tong University.

References

[1] Wei Cai, Xiaoguang Li, and Lizuo Liu. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM Journal on Scientific Computing*, 42(5):A3285–A3312, 2020.

[2] Subrahmanyan Chandrasekhar. *Radiative Transfer*. Courier Corporation, 2013.

[3] MWMG Dissanayake and Nhan Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[4] Weinan E. Machine learning and computational mathematics. *arXiv preprint arXiv:2009.14596*, 2020.

[5] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[6] Weinan E, Jiequn Han, Arnulf Jentzen, et al. Algorithms for solving high dimensional pdes: From nonlinear monte carlo to machine learning. *arXiv preprint arXiv:2008.13333*, 2020.

[7] Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.
[8] Yuwei Fan, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical matrices. *Multiscale Modeling & Simulation*, 17(4):1189–1213, 2019.
[9] A Hamilton, T Tran, MB Mckay, B Quiring, and PS Vassilevski. Dnn approximation of nonlinear finite element equations. Technical report, Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), 2019.
[10] Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *Journal of Computational Mathematics*, 38(3):502–527, 2020.
[11] Jacqueline Lenoble. *Radiative Transfer in Scattering and Absorbing Atmospheres: Standard Computational Procedures*, volume 300. A. Deepak Hampton, Va., 1985.
[12] Xi-An Li, Zhi-Qin John Xu, and Lei Zhang. A multi-scale dnn algorithm for nonlinear elliptic equations with multiple scales. *Communications in Computational Physics*, 28(5):1886–1906, 2020.
[13] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
[14] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.
[15] Yulei Liao and Pingbing Ming. Deep nitsche method: Deep ritz method with essential boundary conditions. *Communications in Computational Physics*, 29(5):1365–1384, 2021.
[16] Ziqi Liu, Wei Cai, and Zhi-Qin John Xu. Multi-scale deep neural network (MscaleDNNs) for solving Poisson-Boltzmann equation in complex domains. *Communications in Computational Physics*, 28(5):1970–2001, 2020.
[17] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.
[18] Stefano Markidis. Physics-informed deep-learning for scientific computing. *arXiv preprint arXiv:2103.09655*, 2021.
[19] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.
[20] Carlos Michelen Strofer, Jin-Long Wu, Heng Xiao, and Eric Paterson. Data-driven, physics-based feature extraction from fluid flow fields using convolutional neural networks. *Communications in Computational Physics*, 25(3):625–650, 2019.
[21] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.
[22] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *arXiv preprint arXiv:2103.10974*, 2021.
[23] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384(C), 2021.

[24] Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *Communications in Computational Physics*, 28(5):1746–1767, 2020.

[25] Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training Behavior of Deep Neural Network in Frequency Domain. In *Neural Information Processing*, Lecture Notes in Computer Science, pages 264–274, 2019.