Journal of Computational Physics 455 (2022) 110996

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Meta-MgNet: Meta multigrid networks for solving parameterized partial differential equations

Yuyan Chen [a], Bin Dong [b,*], Jinchao Xu [c]

[a] School of Mathematical Science, Peking University, China
[b] Beijing International Center for Mathematical Research & Institute for Artificial Intelligence, Peking University, China
[c] Department of Mathematics, Pennsylvania State University, United States of America

ARTICLE INFO

Article history:
Received 1 November 2020
Received in revised form 28 December 2021
Accepted 15 January 2022
Available online 20 January 2022

Keywords:
Parameterized PDEs
Multigrid
Deep-learning
MgNet
Meta-learning

ABSTRACT

This paper studies numerical solutions for parameterized partial differential equations (PDEs) with deep learning. Parametrized PDEs arise in many important application areas, including design optimization, uncertainty analysis, optimal control, and inverse problems. The computational cost associated with these applications using traditional numerical schemes can be exorbitant, especially when the parameters fall into a particular range, and the underlying PDE model is required to be solved with high accuracy using a fine spatial-temporal mesh. Recently, solving PDEs with deep learning has become an emerging field in scientific computing. Existing works demonstrate great potentials of the deep learning-based approach in speeding up numerical solutions of various types of PDEs. However, there is still limited research on the deep learning approach for solving parameterized PDEs. If we directly apply existing deep supervised learning models to solving parameterized PDEs, the models need to be constantly fine-tuned or retrained when the parameters of the PDE change. This limits the applicability and utility of these models in practice. To resolve this issue, we propose a meta-learning based method that can efficiently solve parameterized PDEs with a wide range of parameters without retraining. Our key observation is to regard training a solver for the parameterized PDE with a given set of parameters as a learning task. Then, training a solver for the parameterized PDEs with varied parameters can be viewed as a multi-task learning problem, to which meta-learning is one of the most effective approaches. This new perspective can be applied to many existing PDE solvers to make them suitable for solving parameterized PDEs. As an example, we adopt the Multigrid Network (MgNet) [21] as the base solver. To achieve multi-task learning, we introduce a new hypernetwork, called Meta-NN, in MgNet and refer to the entire network as the Meta-MgNet. Meta-NN takes the differential operators and the right-hand-side of the underlying parameterized PDEs as inputs and generates appropriate smoothers for MgNet, which are essential ingredients for multigrid methods and can significantly affect the convergent speed. The proposed Meta-NN is carefully designed so that Meta-MgNet has guaranteed convergence for Poisson's equation. Finally, extensive numerical experiments demonstrate that Meta-MgNet is more efficient in solving parameterized PDEs than the MG methods and MgNet trained by supervised learning.

* Corresponding author.
E-mail addresses: chenyuyan@pku.edu.cn (Y. Chen), dongbin@math.pku.edu.cn (B. Dong), xu@math.psu.edu (J. Xu).

https://doi.org/10.1016/j.jcp.2022.110996
0021-9991/

1. Introduction

Partial differential equations are essential tools in many areas, such as physics, chemistry, biology, and economics. Most PDEs we encounter in practice contain parameters representing the system's physical or geometric properties, e.g., kinetic characteristics, material properties, the shape of the domain, etc. In practice, we often found ourselves in multi-query scenarios where the PDEs need to be solved for numerous different parameters with high accuracy and efficiency. Such scenarios include design optimization, optimal control, uncertainty quantification, inverse problems, etc. Therefore, a uniformly efficient solver for all parameters is urgently needed.

In this paper, we consider the following parameterized steady-state PDEs

$$\begin{cases} \underset{\sim}{\mathcal{A}_\eta}\, \underset{\sim}{u} = \underset{\sim}{f}, & \text{in } \Omega, \\ \underset{\sim}{u} = \underset{\sim}{u_b}, & \text{on } \partial\Omega, \end{cases} \tag{1}$$

where $\Omega \subset \mathbb{R}^d, d, n \in \mathbb{N}_+$, $\mathfrak{U}$, $\mathfrak{F}$ are two function spaces on Ω, and $\mathfrak{U}_b$ is a function space on $\partial\Omega$, $\underset{\sim}{u} = (u^1, u^2, ..., u^n) \in \mathfrak{U}^n$, $\underset{\sim}{f} = (f^1, f^2, ..., f^n) \in \mathfrak{F}^n$, $\underset{\sim}{u_b} = (u_b^1, u_b^2, ..., u_b^n) \in \mathfrak{U}_b^n$. And $\mathcal{A}_\eta : \mathfrak{U}^n \to \mathfrak{F}^n$ is a linear differential operator with parameter $\eta = (\eta_1, \ldots, \eta_m)$. For convenience, we will omit η when there is not any confusion. In this paper, the specific PDEs we choose for our numerical experiments are 2D/3D steady-state anisotropic diffusion equations and Ossen equations. The steady-state diffusion equations are widely used in fluid mechanics, electronic science, image processing, etc. The Ossen equations play an important role in fluid mechanics. We recall these PDEs as follows:

(1) 2D anisotropic diffusion equations:

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases}$$

where $C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$ is a 2×2 matrix, $\epsilon < 1, \theta \in [0, \pi]$.

(2) 3D anisotropic diffusion equations:

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases}$$

on domain $\Omega = [0, 1]^3$. In this paper, we only concern the case with

$$C = \begin{pmatrix} \epsilon_0 & & \\ & \epsilon_1 & \\ & & \epsilon_2 \end{pmatrix}, \quad \epsilon_0, \epsilon_1, \epsilon_2 > 0.$$

(3) Ossen equations:

$$\begin{cases} -\mu \Delta \underset{\sim}{u} + (\underset{\sim}{a} \cdot \nabla)\underset{\sim}{u} + \nabla p = \underset{\sim}{f}, & \text{in } \Omega, \\ -\text{div}\underset{\sim}{u} = \underset{\sim}{0}, & \text{in } \Omega, \\ \underset{\sim}{u} = \underset{\sim}{0}, & \text{on } \partial\Omega. \end{cases}$$

where $\mu = \dfrac{1}{Re}$ and Re is Reynold number, and $\underset{\sim}{a} = (a_x, a_y)^\top$.

When discretized, equation (1) becomes a linear system of equations

$$\mathbf{A}_\eta \mathbf{u} = \mathbf{f}. \tag{2}$$

Linear system (2) is usually of a very large scale in practice, and iterative methods are often used to solve it. The multigrid (MG) method [49,50,19] is one of the most compelling classical methods. The computational complexity of the MG method for the Poisson's equation is only $O(n)$, where n is the size of the matrix; as compared to another popular high-performance numerical method, the spectral method, whose complexity is $O(n\log n)$. However, the MG method still has trouble solving PDEs (1) with η falling into a specified range. Taking the 2D anisotropic diffusion equation as an example [52], the computational cost of the MG method grows rapidly with $\epsilon \to 0$ (see **Fig. 1**).

Therefore, people try to adjust the parameters and components in the MG method (such as smoothers, prolongations, and restrictions) to improve its performance according to η. Although decades of continuing researches are devoted to the convergence theory of the MG method to find the theoretically best parameters, the computational cost of finding such optimal parameters for a given η can be much higher than solving the linear system (2) itself. For example, [51]

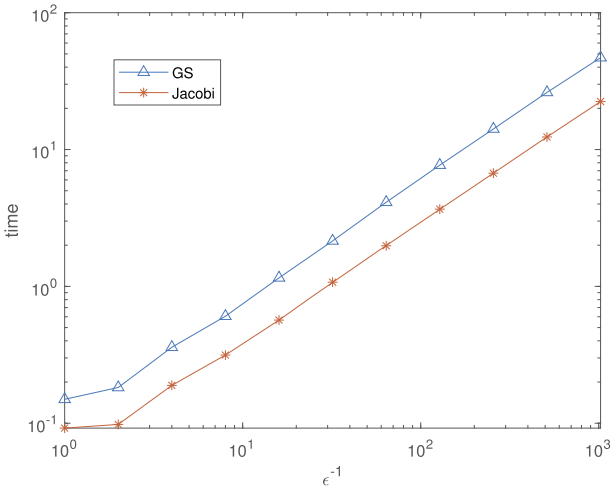

Fig. 1. The wall times of convergence grows rapidly with $\epsilon \to 0$.

derives the theoretically optimal prolongation for a given smoother. However, it requires solving an eigenvalue problem. Crucial parameters such as the damped coefficient of damped Jacobi smoother and the prolongations and restrictions of the algebraic MG method are mostly designed by human experts. However, these manually designed parameters can be rather complicated and have limited improvements for MG methods in practice.

Before the rise of deep learning, machine learning had a similar challenge as MG methods. Classical machine learning heavily relies on feature engineering, where people tried to manually design various types of features that are later fed into a classification or regression model. However, the quality of the features depends on the data set, the downstream task, and also the choice of the classification and regression model. It is extremely difficult to design fully adaptive and good feature extractors purely based on the human experience. This is where deep learning has been proven tremendously effective. After the advent of deep learning, feature extractors can be learned directly from data [28] in an end-to-end fashion. This often leads to feature extractors significantly surpass the previous ones designed by human experts in performance. This enables deep learning to achieve enormous success in many areas of artificial intelligence, such as natural language processing [23] and games [43].

The success of deep learning motivated us to resort to a data-driven approach to determine MG methods' parameters. Among all deep learning approaches, deep supervised learning has become one of the most popular data-driven approaches in scientific computing. Therefore, using deep supervised learning to determine the parameters of the MG method is a natural choice. Since classical MG methods do not work well for certain η, we can learn the parameters in the MG method to make it uniformly efficient for all η.

Deep supervised learning can be easily applied to the MG method if we cast the MG method into a similar formulation as deep neural networks. As first observed in [21], the MG method is an iterative method which can be viewed as a feedforward network. Furthermore, the prolongations, restrictions, and some special smoothers can all be written as convolutions. Therefore, the MG method has a natural connection with the convolutional neural network (CNN). With that, [21] introduced the Multigrid Network (MgNet).

The original MgNet in [21] was proposed for image classification. In this paper, we convert it into a form suitable to solve PDEs and refer to it as PDE-MgNet. PDE-MgNet takes the right-hand-side function $\underset{\sim}{f}$ as input and the approximated solution $\underset{\sim}{u}$ as output. PDE-MgNet performs very well when it is trained on a data set generated by $\eta \sim \mathcal{P}$ in PDEs (1) and tested on some other η generated from the same distribution $\mathcal{P}$. However, it may generalize poorly (i.e., convergence slows down significantly) beyond the training setting, i.e., for $\eta \sim \mathcal{P}'$ with $\mathcal{P}'$ different from $\mathcal{P}$.

This problem is common for deep supervised learning. To resolve it, we need to adopt a more robust learning mechanism. In this paper, we regard learning a solver for PDEs (1) with a given η as one learning task. Then learning solvers for PDEs (1) for all η can be naturally viewed as a multi-task learning problem. In the area of artificial intelligence, meta-learning is an effective approach to solve multi-task learning problems. Thus we propose to use the meta-learning approach to improve PDE-MgNet. Note that PDE-MgNet is just an example we choose in this paper. A similar methodology can be applied to enhance other numerical solvers for parameterized PDEs.

First, let us briefly review meta-learning. Meta-learning [45,24], also known as learning to learn, is a science of studying how different machine learning approaches perform on a wide range of tasks and use these experiences to speed up the learning of new tasks. In contrast, supervised learning trains a model M_η for each task T_η separately. Even though all M_η have the same network structures, we need to retrain the model each time η changes. If there are numerous tasks to learn, the computation cost can be unbearable for supervised learning. The meta-learning approach resolves this problem

by gaining experience over multiple learning episodes - often covering the distribution of related tasks - and uses this experience to improve its future learning performance.

There are several strategies in meta-learning. We briefly introduce two strategies that are suitable for our task of interest. Suppose the only difference between M_η is the weights of neural networks, namely

$$\mathrm{M}_\eta = \mathrm{M}(\mathbf{x}; \mathbf{w}_\eta),$$

where $\mathbf{x}$ is the input and $\mathbf{w}_\eta$ is the weights of neural networks. The two strategies to quickly find suitable $\mathbf{w}_\eta$ for each task η are given as follows.

(1) Finding a good initialization [13,37]: This strategy makes use of a series of tasks $\mathrm{T}_{\eta_1}, \mathrm{T}_{\eta_2}, ..., \mathrm{T}_{\eta_n}$ to obtain a good initial weights $\mathbf{w}_0$ of the deep neural network M. Then, we can easily get $\mathbf{w}_\eta$ for each task η by fine-tuning from $\mathbf{w}_0$. For example, suppose the task T_η is sampled from distribution $\mathcal{P}(\mathrm{T})$, the loss of model M on the task T is $L_{\mathrm{T}}(M)$, and we use gradient descent with learning rate α to train the model. After one step of gradient descent starting from $\mathbf{w}_0$, we obtain the updated weights as $\mathbf{w}' = \mathbf{w}_0 - \alpha \nabla_{\mathbf{w}} L_{\mathrm{T}_\eta}(\mathrm{M}_\eta(\cdot; \mathbf{w}_0))$. Thus, the initial weights $\mathbf{w}_0$ should minimize the following expectation

$$\mathop{\mathbb{E}}_{\mathrm{T}_\eta \sim \mathcal{P}(\mathrm{T})} L_{\mathrm{T}_\eta}(\mathrm{M}_\eta(\cdot; \mathbf{w}')). \tag{3}$$

This is the main idea of Model-Agnostic Meta-Learning (MAML) proposed by [32], where an algorithm solving (3) is also proposed to estimate $\mathbf{w}_0$.

(2) Hypernetwork [17,33,5,54]: This strategy relies on the designs of a network called hypernetwork to infer $\mathbf{w}_\eta$ instead of direct learning of $\mathbf{w}_\eta$ by training. The hypernetwork takes the information on the task η as input and $\mathbf{w}_\eta$ as output. The hypernetwork are trained to make an accurate prediction on $\mathbf{w}_\eta$ for each task η. When the hypernetworks are not powerful enough to make accurate predictions on $\mathbf{w}_\eta$, we treat the approximation of $\mathbf{w}_\eta$ as a good initial value and resort to fine-tuning on each task η to improve the prediction. In particular, if the output of a hypernetwork is the same for all η, we can think that the hypernetwork gives a good initialization for all tasks T_η. In this regard, the previous strategy can be viewed as a special case of the hypernetwork approach.

By viewing solving the parameterized PDE (1) for a given η as a task T_η, we adopt the hypernetwork based meta-learning approach to improve upon the PDE-MgNet. With this, we introduce a new model called Meta-MgNet. Compared to PDE-MgNet, the Meta-MgNet uses a carefully designed hypernetwork to infer the model parameters of the MgNet, instead of learning it directly from data. The hypernetwork grants the Meta-MgNet great ability of in-distribution generalization and out-of-distribution transfer. We shall call this hypernetwork Meta-NN. The Meta-NN is used to infer parameters of specific components in the MgNet. In this paper, we select two types of smoother as an example and design the corresponding meta-smoother (i.e., using Meta-NN to infer parameters of the smoother) for the Meta-MgNet. The two types of smoother are the convolution smoother, which is exactly what MgNet uses, and the smoother based on subspace correction. For the convolution smoother, the Meta-NN infers its convolution kernels. For the smoother based on subspace correction, the Meta-NN infers the spanning vectors of the subspace. The parameters of the Meta-NN are first trained on a data set with mixed data from different η. Then, we can fine-tune the Meta-NN while solving a specific T_η. However, our numerical experiments show that the weights given by Meta-NN without retraining are good enough. Therefore, we shall skip the retraining step in the experiments.

1.1. Related work

In the area of solving PDEs by machine learning, especially deep learning, the existing algorithms can be divided into the following two categories.

(1) Using neural networks to approximate the function $\underset{\sim}{u}$: These algorithms are suitable for a PDE with a fixed differential operator $\mathcal{A}$ and the right-hand side $\underset{\sim}{f}$. The most notable advantage of this approach is that they can: 1) overcome the curse of dimensionality and solve high-dimension PDEs; 2) resolve complex geometries in the solution due to the mesh-less representation of neural networks. Successful examples include the nonlinear convection-diffusion PDEs [39,40], Riemann Problem [36], high-dimension PDEs [3,11,20,44], and others [47,53,46,31,42]. Nevertheless, if the parameter η or the right-hand-side function $\underset{\sim}{f}$ is changed, the neural network often needs to be retrained.

(2) Using neural networks to approximate the solution operator $\mathcal{A}^{-1}$: The general modeling strategy of methods in this category is to replace a portion of the classical numerical method with neural networks to improve its performance. For example, [41] uses NN to estimate locations of discontinuous, [9] uses NN to introduce an appropriate amount of artificial dissipation in the numerical solver. There are also works using NN to approximate the entire operator $\mathcal{A}^{-1}$. For example, [25] trains a U-Net as a solver for Poisson's equations. The most related work to the current one is [12], where the authors use a meta-learning approach for parameterized pseudo-differential operators with deep neural networks.

However, there are two main differences between their work and ours. Firstly, their idea is to find the map $\eta \mapsto \mathcal{A}_\eta^{-1}$ directly. Ours is based on the observation that learning solvers for PDE (1) with different η is a multi-task learning problem and then introduce meta-learning to solve it. Secondly, their approach is based on the wavelet method, while ours is based on the MgNet.

There are also several studies on improving the MG method by deep-learning. For instance, [27,15,35] proposed a series of data-driven approaches to design prolongations and restrictions in MG. However, these methods are based on supervised learning while we focus on learning smoothers based on meta-learning.

In addition to the approaches above, the reduced-order modeling (ROM) [1,4] is also a widely used method for solving parameterized PDEs. The objective of ROM is to generate reduced models that are cheaper to solve while still well approximate the original PDEs. For example, [29] proposes a novel framework for projecting dynamical systems onto nonlinear manifolds using minimum-residual formulations at the time-continuous and time-discrete levels; [14] proposes new deep learning-based nonlinear ROM.

The remaining sections are organized as follows. In section 2, we will introduce necessary notations for the rest of the paper and discuss how to represent discrete PDEs by convolutions. In section 3, we review the multigrid method and the PDE-MgNet. In section 4, we introduce the proposed Meta-MgNet and provide a preliminary convergence analysis of the algorithm. In section 5, we present the numerical experiments and comparisons among the classical MG methods, PDE-MgNet, and Meta-MgNet. Concluding remarks are given in section 6.

2. Convolutions and differential operators

The key to solving PDEs is to discretize the differential operators properly. The main goal of this section is to present a theorem that convolutions can express the most meaningful discretizations of differential operators. This theorem plays an essential role for us to rewrite traditional numerical solvers as CNNs. The MgNet introduced by [21] is an example, which is a reformulation from the MG method. Furthermore, we will introduce the definition of convolution operators and then show how to use convolutions to represent discrete forms of differential operators.

2.1. Convolution operators

In this paper, we only consider the convolution of two second-order tensors and the convolution of a fourth-order tensor and a third-order tensor. Consider two second order tensors $\mathsf{K} = (\mathsf{K}_{\jmath,\imath})$, with $\jmath, \imath \in \mathbb{Z}$ and $\mathsf{v} = (\mathsf{v}_{j,i})$, with $i \in \{0, 1, ..., I\}$, $j \in \{0, 1, ..., J\}$, i.e.

$$\mathsf{K} = \begin{pmatrix} \ddots & & \vdots & & \iddots \\ & \mathsf{K}_{-1,-1} & \mathsf{K}_{-1,0} & \mathsf{K}_{-1,1} & \\ \cdots & \mathsf{K}_{0,-1} & \mathsf{K}_{0,0} & \mathsf{K}_{0,1} & \cdots \\ & \mathsf{K}_{1,-1} & \mathsf{K}_{1,0} & \mathsf{K}_{1,1} & \\ \iddots & & \vdots & & \ddots \end{pmatrix} \quad \text{and} \quad \mathsf{v} = \begin{pmatrix} \mathsf{v}_{0,0} & \mathsf{v}_{0,1} & \cdots & \mathsf{v}_{0,I} \\ \mathsf{v}_{1,0} & \mathsf{v}_{1,1} & \cdots & \mathsf{v}_{1,I} \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{v}_{J,0} & \mathsf{v}_{J,1} & \cdots & \mathsf{v}_{J,I} \end{pmatrix}.$$

The convolution $\mathsf{K} \star \mathsf{v}$ is also a second-order tensor, and the values of its components are

$$(\mathsf{K} \star \mathsf{v})_{j,i} = \sum_{\jmath,\imath \in \mathbb{Z}} \mathsf{K}_{\jmath,\imath} \mathsf{v}_{j+\jmath,i+\imath},$$

where $i \in \{0, 1, ..., I\}$, $j \in \{0, 1, ..., J\}$. If $i + \imath \notin \{0, 1, ..., I\}$ or $j + \jmath \notin \{0, 1, ..., J\}$, we set $\mathsf{v}_{j+\jmath,i+\imath} = 0$.

Consider a forth-order tensor K and a third-order tensor v, where $\mathsf{K} = (\mathsf{K}_{l,k,\jmath,\imath})$, with $l, k \in \{1, 2, ..., S\}$, $\jmath, \imath \in \mathbb{Z}$ and $\mathsf{v} = (\mathsf{v}_{k,j,i})$, with $k \in \{1, 2, ..., S\}$, $i \in \{0, 1, ..., I\}$, $j \in \{0, 1, ..., J\}$. The definitions of convolution, convolution with stride and deconvolution (the transpose of convolution) are listed below

(1) **Convolution**: The convolution $\mathsf{K} \star \mathsf{v}$ is a third order tensor and the value of its components are

$$(\mathsf{K} \star \mathsf{v})_{l,j,i} = \sum_{k=1}^{S} \sum_{\jmath,\imath \in \mathbb{Z}} \mathsf{K}_{l,k,\jmath,\imath} \mathsf{v}_{k,j+\jmath,i+\imath}.$$

(2) **Convolution with strides**: Suppose $I_s, J_s \in \mathbb{N}^+$, and we use $\star_{J_s,I_s}$ to express the convolution with stride$= (J_s, I_s)$ and the components of $\mathsf{K} \star_{J_s,I_s} \mathsf{v}$ are

$$(\mathsf{K} \star_{J_s,I_s} \mathsf{v})_{l,j,i} = \sum_{k=1}^{S} \sum_{\jmath,\imath \in \mathbb{Z}} \mathsf{K}_{l,k,\jmath,\imath} \mathsf{v}_{k,jJ_s+\jmath,iI_s+\imath}.$$

If $I_s = J_s$, we can write $\star_{J_s,I_s}$ briefly as $\star_{J_s}$.

(3) **Deconvolution**: Suppose I_s, $J_s \in \mathbb{N}^*$, and we use $\star^{J_s,I_s}$ to express deconvolution with strides$=(J_s, I_s)$ and the components of $\mathsf{K}\star^{J_s,I_s}\mathsf{v}$ are

$$(\mathsf{K}\star^{J_s,I_s}\mathsf{v})_{l,jJ_s+j',iI_s+i'} = \sum_{k=1}^{S}\sum_{\jmath,\imath\in\mathbb{Z}} \mathsf{K}_{l,k,\jmath J_s+j',\imath I_s+i'}\mathsf{v}_{\kappa,j+\jmath,i+\imath}.$$

If $I_s = J_s$, we can write $\star^{J_s,I_s}$ briefly as $\star^{J_s}$. We can also regard deconvolution as a convolution after upsampling. For example, we have

$$\mathsf{K}\star(\mathsf{v}\uparrow) = \mathsf{K}\star^2\mathsf{v}.$$

2.2. Representing discretized PDEs in convolutions

We discuss how to transform the $\mathcal{A}_\eta$ to the convolution form. We first write components of (1) in the following matrix form

$$\begin{pmatrix} \mathcal{A}_{1,1} & \mathcal{A}_{1,2} & \cdots & \mathcal{A}_{1,n} \\ \mathcal{A}_{2,1} & \mathcal{A}_{2,2} & \cdots & \mathcal{A}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{A}_{n,1} & \mathcal{A}_{n,2} & \cdots & \mathcal{A}_{n,n} \end{pmatrix} \begin{pmatrix} u^1 \\ u^2 \\ \vdots \\ u^n \end{pmatrix} = \begin{pmatrix} f^1 \\ f^2 \\ \vdots \\ f^n \end{pmatrix}, \tag{4}$$

where each linear differential operator $\mathcal{A}_{i,j}(i, j = 1, 2, ..., n)$ is a component of $\mathcal{A}_\eta$. Thanks to the linear superposition property of each component in (4), we only need to consider each component separately, i.e., for given linear differential operator $\mathcal{K}$ and a functional f in $\mathfrak{V}'$, find $v \in \mathfrak{V}$ to satisfy

$$\mathcal{K}v = f. \tag{5}$$

Our goal is to find a kernel K, and tensors v and f to represent the discretization of (5) as

$$\mathsf{K}\star\mathsf{v} = \mathsf{f}. \tag{6}$$

Suppose that the Galerkin method, e.g. the finite difference method (FDM) or finite element method (FEM), is used to discretize the PDEs (5). Now, we propose a general way to convert (5) into the convolution form (6). According to the Galerkin method, we first convert PDEs (5) to its weak form:

$$\text{find } v \in \mathfrak{V}, \text{such that } \forall w \in \mathfrak{V}, K(v, w) = f(w). \tag{7}$$

Then, we choose a finite dimensional subspace $\mathfrak{V}_h \subset \mathfrak{V}$ to discretize (7) and solve the projected problem:

$$\text{find } v_h \in \mathfrak{V}_h, \text{ such that } \forall w_h \in \mathfrak{V}_h, K(v_h, w_h) = f(w_h).$$

Let Φ be a set of basis of $\mathfrak{V}_h$ and assume that it satisfies the following assumptions.

Assumption 2.1. *Suppose Φ can be divided into S groups $\Phi_1, ..., \Phi_S$, where $\Phi_k = \{\phi_{k,j,i} | i = 0, 1, 2..., I_k, j = 0, 1, 2, ..., J_k\}$ and each Φ_k can be generated by translations along the grid-lines from a compact support function φ_k, i.e. $\exists\varphi_k\ \forall i \in \{0, 1, 2, ..., I_k\}, j \in \{0, 1, 2, ..., J_k\}$ such that*

$$\phi_{k,j,i}(x, y) = \varphi_k(x - ih, y - jh).$$

Then, we have the following theorem stating that the PDE (5) discretized by FDM or FEM can be expressed in convolution form.

Theorem 1. *If a discretized scheme in FDM or FEM satisfies Assumption 2.1, the discretized PDE $\mathcal{K}v = f$ can be written in the form $\mathsf{K}\star\mathsf{v} = \mathsf{f}$ with K and f given as follows*

$$\mathsf{K} = (\mathsf{K}_{l,k,j,i}) = K(\varphi_k(x - ih, y - jh), \varphi_l(x, y)) \quad \text{and} \quad \mathsf{f} = (\mathsf{f}_{l,j,i}) = f(\phi_{l,j,i}). \tag{8}$$

Proof. With Assumption 2.1, we can write v_h as

$$v_h = \sum_{k=1}^{S}\sum_{\jmath=0}^{J_k}\sum_{\imath=0}^{I_k} \mathsf{v}_{k,\jmath,\imath}\phi_{k,\jmath,\imath}.$$

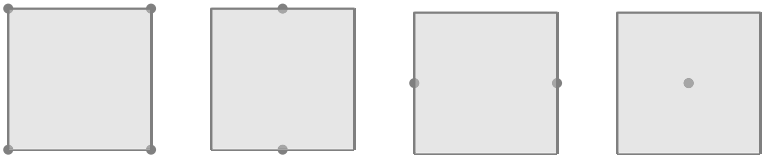

Fig. 2. Different case of point-wise discretion.

According to Galerkin method, we have

$$\begin{aligned}
K(v_h, \phi_{l,j,i}) &= K(\sum_{k=1}^{S}\sum_{\jmath=0}^{J_k}\sum_{\imath=0}^{I_k} \mathsf{v}_{k,\jmath,\imath}\phi_{k,\jmath,\imath}, \phi_{l,j,i}) = \sum_{k=1}^{S}\sum_{\jmath=0}^{J_k}\sum_{\imath=0}^{I_k} \mathsf{v}_{k,\jmath,\imath} K(\phi_{k,\jmath,\imath}, \phi_{l,j,i}) \\
&= \sum_{k=1}^{S}\sum_{\jmath=0}^{J_k}\sum_{\imath=0}^{I_k} \mathsf{v}_{k,\jmath,\imath} K(\varphi_k(x-\imath h, y-\jmath h), \varphi_l(x-ih, y-jh)) \\
&= \sum_{k=1}^{S}\sum_{\jmath=-j}^{J_k-j}\sum_{\imath=-i}^{I_k-i} \mathsf{v}_{k,\jmath+j,\imath+i} K(\varphi_k(x-(\imath+i)h, y-(\jmath+j)h), \varphi_l(x-ih, y-jh)) \\
&= \sum_{k=1}^{S}\sum_{\jmath=-j}^{J_k-j}\sum_{\imath=-i}^{I_k-i} \mathsf{v}_{k,\jmath+j,\imath+i} K(\varphi_k(x-\imath h, y-\jmath h), \varphi_l(x, y)).
\end{aligned}$$

Let $\mathsf{K} = (\mathsf{K}_{l,k,j,i})$, where

$$\mathsf{K}_{l,k,j,i} = K(\varphi_k(x-ih, y-jh), \varphi_l(x, y)). \tag{9}$$

While $i \notin \{0, 1, ..., I_k\}$ or $j \notin \{0, 1, ..., J_k\}$, we set $\mathsf{v}_{k,j,i} = 0, \mathsf{K}_{l,k,j,i} = 0$. Then, we have

$$K(v_h, \phi_{l,j,i}) = \sum_{k=1}^{S}\sum_{\jmath=-j}^{J_k-j}\sum_{\imath=-i}^{I_k-i} \mathsf{v}_{k,\jmath+j,\imath+i}\mathsf{K}_{l,k,\jmath,\imath} = \sum_{k=1}^{S}\sum_{\jmath\in\mathbb{Z}}\sum_{\imath\in\mathbb{Z}} \mathsf{v}_{k,\jmath+j,\imath+i}\mathsf{K}_{l,k,\jmath,\imath} = (\mathsf{K}\star\mathsf{v})_{l,j,i}.$$

Let $\mathsf{f} = (\mathsf{f}_{l,j,i})$ with $\mathsf{f}_{l,j,i} = f(\phi_{l,j,i})$. We obtain

$$K(v_h, \phi_{l,j,i}) = f(\phi_{l,j,i}), \quad \forall l = 1, 2, ..., S,$$

which is the same as (6). □

Although we have demonstrated a generic method to convert PDEs (5) into the convolution form, it is sometimes inconvenient to use. We can calculate K in an easier way for most discretization schemes, as will be described in the remaining part of this subsection. For both FDM and FEM discretization, we assume that the mesh $\mathcal{T}$ is $N \times N$ uniform triangular or rectangular mesh, and let $h = \dfrac{1}{N}$.

2.2.1. Finite difference methods (FDM)

FDM is one of the most popular discretizations used for solving PDEs by approximating them with difference equations. The basis functions of FDM are Legendre polynomials. Thus, it is not convenient to use (8) to calculate K and f. Fortunately, we can use Taylor's expansion to compute entries of K and f. Furthermore, the functions v and f can be easily discretized by using their values restricted on the mesh $\mathcal{T}$. As examples, we present four commonly seen cases as follows (see **Fig. 2**).

(1) Vertex of an element: set $\mathsf{v}_{j,i} = v(ih, jh),\ i, j = 0, 1, ..., N$;
(2) Midpoint of a horizontal edge: set $\mathsf{v}_{j,i} = v((i-0.5)h, jh),\ i = 1, 2, ..., N, j = 0, 1, ..., N$;
(3) Midpoint of a vertical edge: set $\mathsf{v}_{j,i} = v(ih, (j-0.5)h),\ i = 0, 1, ..., N, j = 1, 2, ..., N$;
(4) Center of an element, set $\mathsf{v}_{j,i} = v((i-0.5)h, (j-0.5)h),\ i, j = 1, 2, ..., N$.

After choosing a discretization for v and f, we use Taylor expansion to discretize $\mathcal{K}$. Suppose $p \in \mathbb{N}, \boldsymbol{\alpha} = (\alpha_1, \alpha_2), |\boldsymbol{\alpha}| = \alpha_1 + \alpha_2, \partial^{\boldsymbol{\alpha}} = \dfrac{\partial^{\alpha_1+\alpha_2}}{(\partial x)^{\alpha_1}(\partial y)^{\alpha_2}}$ and $\mathcal{K} = \sum_{|\boldsymbol{\alpha}|\leqslant p} a_{\boldsymbol{\alpha}}\partial^{\boldsymbol{\alpha}}$. Then, we can obtain the kernel K from a difference scheme. If $\forall\boldsymbol{\alpha}$ such that $\boldsymbol{\alpha} \leqslant p$, we have a finite difference approximation of $\partial^{\boldsymbol{\alpha}}$ as $\partial_h^{\boldsymbol{\alpha}}\mathsf{v}_{j,i} \approx \sum_{\jmath,\imath} q_{\jmath,\imath}^{\boldsymbol{\alpha}}\mathsf{v}_{j+\jmath,i+\imath}$ and the expression of each

Table 1
Some common finite difference schemes and its corresponding kernels K.

$\mathcal{K}$	Difference scheme	Kernel K
∂_x	$\frac{1}{h}(v_{i,j} - v_{i-1,j})$	$\frac{1}{h}\begin{pmatrix}-1 & 1 & 0\end{pmatrix}$
	$\frac{1}{h}(v_{i+1,j} - v_{i,j})$	$\frac{1}{h}\begin{pmatrix}0 & -1 & 1\end{pmatrix}$
∂_y	$\frac{1}{h}(v_{i,j} - v_{i,j-1})$	$\frac{1}{h}\begin{pmatrix}-1\\1\\0\end{pmatrix}$
	$\frac{1}{h}(v_{i,j+1} - v_{i,j})$	$\frac{1}{h}\begin{pmatrix}0\\-1\\1\end{pmatrix}$
∂_{xx}	$\frac{1}{h^2}(v_{i-1,j} + v_{i+1,j} - 2v_{i,j})$	$\frac{1}{h^2}\begin{pmatrix}1 & -2 & 1\end{pmatrix}$
∂_{xy}	$\frac{1}{4h^2}(v_{i-1,j-1} + v_{i+1,j+1} - v_{i+1,j-1} - v_{i-1,j+1})$	$\frac{1}{4h^2}\begin{pmatrix}1 & 0 & -1\\0 & 0 & 0\\-1 & 0 & 1\end{pmatrix}$
∂_{yy}	$\frac{1}{h^2}(v_{i,j-1} + v_{i,j+1} - 2v_{i,j})$	$\frac{1}{h^2}\begin{pmatrix}1\\-2\\1\end{pmatrix}$
Δ	$\frac{1}{h^2}(v_{i-1,j} + v_{i+1,j} + v_{i,j-1} + v_{i,j+1} - 4v_{i,j})$	$\frac{1}{h^2}\begin{pmatrix}0 & 1 & 0\\1 & -4 & 1\\0 & 1 & 0\end{pmatrix}$

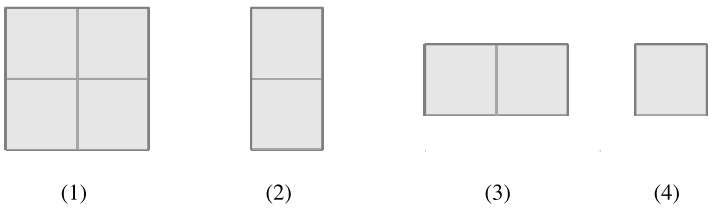

Fig. 3. Some usual support of base function in FEM on rectangular mesh.

component of K is $\mathsf{K}_{j,\iota} = \sum_{|\boldsymbol{\alpha}|\leqslant p}\sum_{j,\iota} a_{\boldsymbol{\alpha}} q_{j,\iota}^{\boldsymbol{\alpha}}$. Common finite difference approximations of partial derivatives and their corresponding convolution kernels are listed in **Table 1**.

2.2.2. Finite element methods (FEM)

FEM is more complicated than FDM because a variable v usually contains several types of basis functions. Each type of basis functions is a channel of the tensor v. For most FEM methods on the rectangle mesh, we can divide each basis functions into the following 4 cases based on the support of the functions, i.e. 2×2, 2×1, 1×2, and 1×1 elements (see **Fig. 3**). These four cases can be reduced to the case for FDM.

On the triangular mesh, each basis function can be divided into 6 cases, which are shown in **Fig. 4** according to the shape of the support of the function. For case (1)∼(4), they can also be reduced to the case of FDM, while for case (5) or (6), we may have to apply **Theorem 1** to calculate K and f.

3. The multigrid method and PDE-MgNet

In this section, we briefly describe the geometric MG method and PDE-MgNet. MG method is one of the most high-efficiency methods for solving PDEs. We consider the discrete form of the parameterized PDEs (2).

3.1. Multigrid method

Iterative method is one of the basic numerical methods for solving the linear system (2). Given an initial guess $\mathbf{u}_0$ and an update scheme represented by $\mathcal{H}$, we can write the iterative method generically as

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \mathcal{H}(\mathbf{u}_t),\ t = 0, 1\ldots, T. \tag{10}$$

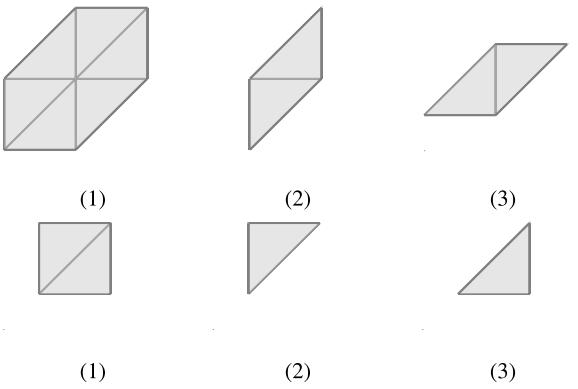

Fig. 4. Some usual support of base function in FEM on triangular mesh.

Note that we can regard (10) as a dynamical system or a feed-forward network. Such perspective is the key to connect deep neural networks (e.g. ResNet [22]) with dynamic systems, PDEs and optimal control [16,8,10,18,34,7].

In addition to the iterative scheme (10), the residual correction scheme

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \mathcal{H}(\mathbf{f} - \mathbf{A}\mathbf{u}_t),\ t = 0, 1 \ldots, T, \tag{11}$$

is one of most important types of iterative method. The MG method is one of such iterative methods, which is written as:

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \mathrm{Mg}(\mathbf{f} - \mathbf{A}\mathbf{u}_t),\ t = 0, 1 \ldots, T, \tag{12}$$

where the Mg operator in (12) is given by **Algorithm 1**.

The MG operator can be divided into two steps, i.e. smoothing and coarse grid correction. The smoothing step is to use a smoother to eliminate high-frequency errors. A smoother $\mathcal{B}$ (or its matrix form $\mathbf{B}$) is often an iterative scheme by itself taking the form

$$\mathbf{u}_+ = \mathbf{u}_0 + \mathbf{B}(\mathbf{f} - \mathbf{A}\mathbf{u}_0).$$

Popular choices of the smoother in MG include the Jacobi and Gauss-Seidel (GS) smoother, which are listed below

$$\mathbf{B} = \begin{cases} \mathrm{diag}(\mathbf{A})^{-1} & \text{Jacobi} \\ \mathrm{tril}(\mathbf{A})^{-1} & \text{GS} \end{cases}. \tag{13}$$

The Jacobi or GS smoother can efficiently eliminate high-frequency approximation errors. However, they are ineffective for low-frequency errors, which is where the coarse grid correction is needed. The MG methods utilize the solution on a coarse grid to approximate the low-frequency error. As a simple example, we illustrate the steps of coarse grid correction in a two-level MG in **Fig. 5**.

For the multi-level MG method, if we have a sequence of grids $\mathcal{T}^1(=\mathcal{T}), \mathcal{T}^2, \ldots, \mathcal{T}^J$ and assume that the prolongation and restriction between $\mathcal{T}^\ell$ and $\mathcal{T}^{\ell+1}$ are $\mathbf{P}_{\ell+1}^{\ell}$ and $\mathbf{R}_{\ell}^{\ell+1}$. Let $\mathbf{A}^{\ell+1} = \mathbf{R}_{\ell}^{\ell+1}\mathbf{A}^{\ell}\mathbf{P}_{\ell+1}^{\ell}$. The **Algorithm 1** presents the algorithm of multi-level MG.

3.2. PDE-MgNet

The original MgNet proposed by [21] is a new CNN model for image classification, which is inspired by the connection between CNNs and the MG method. As observed by [21] that the smoothers, prolongations, and restrictions can be represented by convolutions. Thus, we can write the corresponding kernels as B, P, and R. Therefore, the MG method can be naturally reformulated as a CNN model, which was called the MgNet by [21].

In this paper, we focus on solving PDEs. Thus, we need to modify the original MgNet in [21] to be suitable for solving PDEs. We shall call the modified MgNet the PDE-MgNet. We formulate every component of PDE-MgNet in the convolution form. For that, we consider the convolution form of PDEs (1):

$$\mathsf{A}_\eta \star \mathsf{u} = \mathsf{f}. \tag{14}$$

PDE-MgNet replaces the smoother $\mathcal{B}^\ell$, the prolongation $\mathcal{P}$, the restriction $\mathcal{R}$ and $\mathbf{A}^\ell$ in the MG methods described in **Algorithm 1** with trainable convolution operators. **Figure 6** shows the architecture of a two-level \-Cycle PDE-MgNet and the multi-level case of PDE-MgNet is presented in **Algorithm 2**.

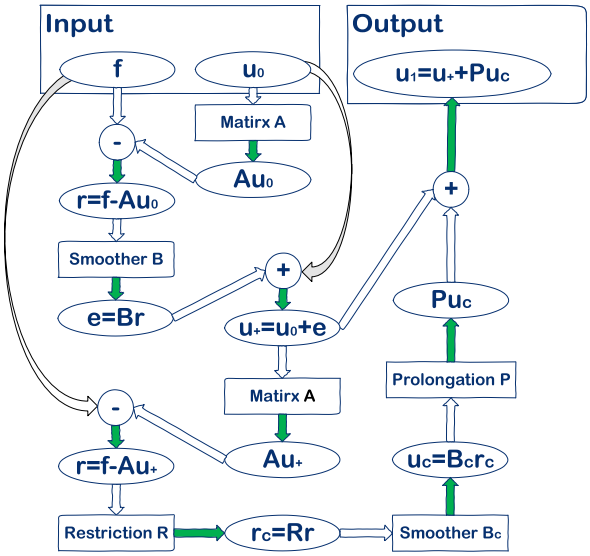

Fig. 5. Two level \-Cycle MG.

Algorithm 1 $\mathbf{u} = \mathrm{Mg}(\mathbf{f}; J, \nu_1, \cdots, \nu_J)$.

Hyper-parameters: number of grids J, times of smooth in each grid: $\nu_1, \cdots, \nu_J$
Input: right hand side $\mathbf{f}$
Output: approximate solution $\mathbf{u}$
Initialization:

$$\mathbf{f}^1 \leftarrow \mathbf{f}, \quad \mathbf{u}^{1,0} \leftarrow \mathbf{0}, \quad \mathbf{r}^{1,0} \leftarrow \mathbf{f}.$$

Smoothing and restriction from fine to coarse level (nested)
for $\ell = 1 : J$ **do**
  Smoothing
  if $\ell = J$ **then**

$$\mathbf{u}^{\ell,1} \leftarrow (\mathbf{A}^\ell)^{-1}\mathbf{r}^{\ell,0}$$

  else
    for $i = 1 : \nu_\ell$ **do**

$$\mathbf{u}^{\ell,i} \leftarrow \mathbf{u}^{\ell,i-1} + \mathbf{B}^\ell \mathbf{r}^{\ell,i-1}$$

$$\mathbf{r}^{\ell,i} \leftarrow \mathbf{f}^\ell - \mathbf{A}^\ell \mathbf{u}^{\ell,i}.$$

    end for
  end if
  Form restricted residual

$$\mathbf{f}^{\ell+1} \leftarrow \mathbf{R}_\ell^{\ell+1}\mathbf{r}^{\ell,\nu_\ell}, \quad \mathbf{u}^{\ell+1,0} \leftarrow \mathbf{0}, \quad \mathbf{r}^{\ell+1,0} \leftarrow \mathbf{f}^{\ell+1}.$$

end for
Prolongation and restriction from coarse to fine level
for $\ell = J - 1 : 1$ **do**
  Coarse grid correction

$$\mathbf{u}^{\ell,\nu_\ell} \leftarrow \mathbf{u}^{\ell,\nu_\ell} + \mathbf{P}_{\ell+1}^\ell \mathbf{u}^{\ell+1,\nu_\ell}.$$

end for
return $\mathbf{u} = \mathbf{u}^{1,\nu_1}$.

With **Algorithm 2**, we obtain the iterative PDE-MgNet for solving (14):

$$\mathsf{u}_{t+1} = \mathsf{u}_t + \text{PDE-MgNet}(\mathsf{f} - \mathsf{A} \star \mathsf{u}_t), \ t = 0, 1 \ldots, T, \tag{16}$$

with $\mathsf{u}_0 = 0$. For convenience, we shall refer to the iterative PDE-MgNet (16) simply as the PDE-MgNet. Note that the PDE-MgNet (16) is precisely the MG method (12) with $\mathcal{A}$, $\mathcal{B}$, $\mathcal{P}$ and $\mathcal{R}$ replaced by convolutions, and it reduces to the original MgNet proposed by [21] when the prolongation step is replaced by a classifier.

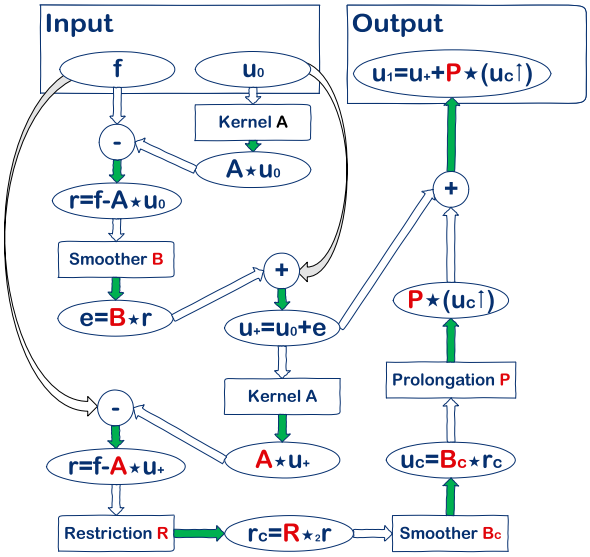

Fig. 6. Two level \-Cycle PDE-MgNet.

Algorithm 2 $\mathsf{u} = \text{PDE-MgNet}(\mathsf{f};\ J, \nu_1, \cdots, \nu_J)$.

Hyper-parameters: number of grids J, times of smooth in each grid: $\nu_1, \cdots, \nu_J$
Input: right-hand side f
Output: approximate solution u
Initialization

$$\mathsf{f}^1 \leftarrow \mathsf{f}, \quad \mathsf{u}^{1,0} \leftarrow 0, \quad \mathsf{r}^{1,0} \leftarrow \mathsf{f}.$$

Smoothing and restriction from fine to coarse level
for $\ell = 1 : J$ **do**
Smoothing:
if $\ell = J$ **then**
Convert $\mathsf{r}^{\ell,0}$ into vector form $\mathbf{r}^{\ell,0}$ and A^ℓ into matrix form $\mathbf{A}^\ell$.

$$\mathbf{u}^{\ell,1} \leftarrow (\mathbf{A}^\ell)^{-1}\mathbf{r}^{\ell,0}.$$

Convert $\mathbf{u}^{\ell,1}$ into tensor form $\mathsf{u}^{\ell,1}$.
else
for $i = 1 : \nu_\ell$ **do**

$$\mathsf{u}^{\ell,i} \leftarrow \mathsf{u}^{\ell,i-1} + \mathsf{B}^{\ell,i-1} \star \mathsf{r}^{\ell,i-1}, \tag{15}$$

$$\mathsf{r}^{\ell,i} \leftarrow \mathsf{f}^\ell - \mathsf{A}^\ell \star \mathsf{u}^{\ell,i}.$$

end for
end if
Form restricted residual

$$\mathsf{f}^{\ell+1} \leftarrow \mathsf{R}_\ell^{\ell+1} \star_2 \mathsf{r}^{\ell,\nu_\ell}, \quad \mathsf{u}^{\ell+1,0} \leftarrow 0, \quad \mathsf{f}^{\ell+1,0} \leftarrow \mathsf{f}^{\ell+1}.$$

end for
Prolongation from coarse to fine level
for $\ell = J - 1 : 1$ **do**
Coarse grid correction

$$\mathsf{u}^{\ell,\nu_\ell} \leftarrow \mathsf{u}^{\ell,\nu_\ell} + \mathsf{P}_{\ell+1}^{\ell} \star^2 \mathsf{u}^{\ell+1,\nu_\ell}.$$

end for
return $\mathsf{u} = \mathsf{u}^{1,\nu_1}$.

The values of the convolution kernels of PDE-MgNet are learned from data by minimizing a loss function defined similarly as in [30]. Suppose the distribution of the right-and-side function f is F. If we sample the distribution F by M_{train} times and denote the training data set as $\mathfrak{X}_F$, then the empirical loss is given by

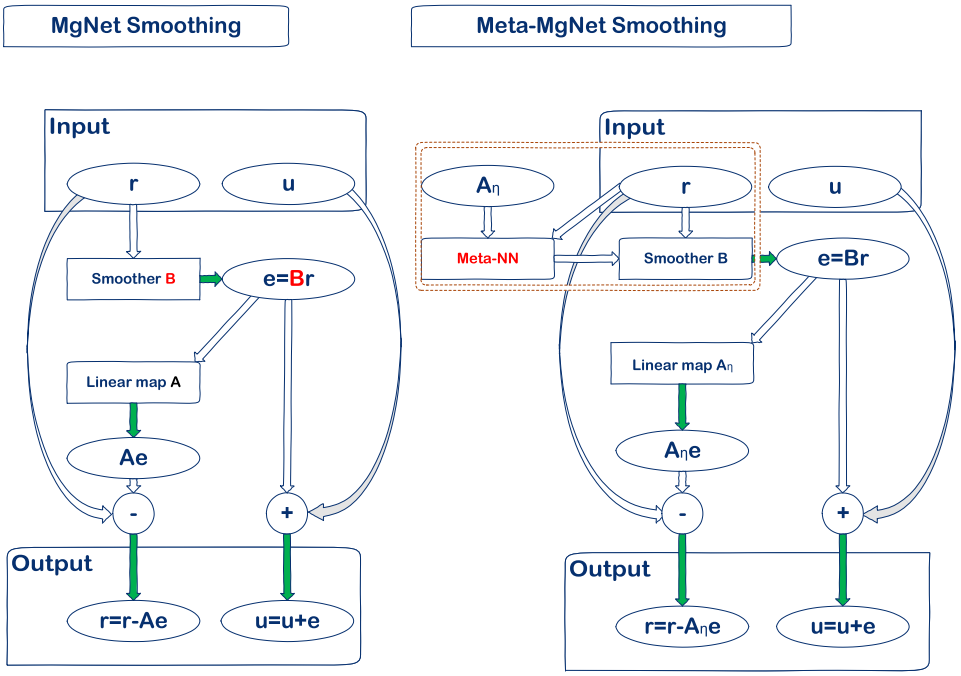

Fig. 7. Comparison between PDE-MgNet and Meta-MgNet in smoothing step. The red rectangle shows the major difference. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$$Loss \approx \frac{1}{M_{\text{train}}} \sum_{\mathsf{f} \in \mathfrak{X}_F} \frac{||\mathsf{f} - \mathsf{A} \star \mathsf{u}_T||^2}{||\mathsf{f}||^2}. \tag{17}$$

In our experiments, we choose $T = 1$ following [27].

4. Meta-MgNet

The PDE-MgNet suffers from poor generalization when tested on a data set generated from the distribution far away from that of the training set, which significantly limits the practicality and utility of PDE-MgNet. This motivates us to improve PDE-MgNet with meta-learning by introducing a properly designed hypernetwork which infers specific components of the PDE-MgNet according to the parameters η of the parameterized PDE to achieve uniformly fast convergence. In this paper, the hypernetwork is introduced to make the smoother $\mathcal{B}$ in PDE-MgNet PDE-dependent. Now, we shall describe details of the design of such hypernetwork and the architecture of the entire Meta-MgNet.

4.1. Architecture of Meta-MgNet

The hypernetwork we introduce to the PDE-MgNet is called Meta-NN. The Meta-MgNet uses Meta-NN to infer an appropriate smoother (called meta-smoother) for each parameter η. The architecture of the meta-smoother in comparison with the smoother of the PDE-MgNet is presented in **Fig. 7**. The advantage of Meta-MgNet over PDE-MgNet is that the smoother of Meta-MgNet changes according to A_η and r, or we can write $\mathcal{B} = \mathcal{B}_{\mathsf{A}_\eta,\mathsf{r}}$ which is realized by the Meta-NN. The entire architecture of the (iterative) Meta-MgNet solving (14) is given by

$$\mathsf{u}_{t+1} = \mathsf{u}_t + \text{Meta-MgNet}(\mathsf{f} - \mathsf{A}_\eta \star \mathsf{u}_t, \mathsf{A}_\eta), \tag{18}$$

where the Meta-MgNet$(\cdot)$ is computed using **Algorithm 2** with (15) replaced by the meta-smoothing:

$$\mathsf{u}^{\ell,i} \leftarrow \mathsf{u}^{\ell,i-1} + \mathcal{B}^{\ell,i-1}_{\mathsf{A}_\eta,\mathsf{r}}(\mathsf{r}^{\ell,i-1}).$$

In this paper, we consider two different methods to realize the meta-smoother $\mathcal{B}_{\mathsf{A}_\eta,\mathsf{r}}$. The first one is based on the convolutional smoother, and its kernels are inferred from a convolutional hypernetwork. Thus, we call it the direct method, which is natural but mediocre. The second one is based on subspace correction smoother. Using subspace correction as a smoother is not as common as other smoothers such as Gauss-Seidel, but the numerical experiments show it performs better than direct methods.

4.1.1. Direct method

In the basic structure of PDE-MgNet, the smoothers are convolutions. Therefore, the direct method is to use Meta-NN to infer the value of these convolution kernels. We can use a vanilla DNN as the Meta-NN. **Algorithm 3** presents the details of this method. As for the structure of Meta-NN, we use a fully connect neural network with two hidden layers with 100 neurons in each layer.

Algorithm 3 $\mathcal{B} = B_{\mathrm{d}}(\mathsf{r}, \mathsf{A}_\eta; \mathcal{G})$.

Hyper-parameters: $\mathcal{G}$
Inputs: r, A_η,
Outputs: $\mathcal{B}$
1. Calculate subspace:

$$\mathsf{B} \leftarrow \mathcal{G}(\frac{\mathsf{r}}{||\mathsf{r}||}, \mathsf{A}_\eta).$$

2. Define the effect of $\mathcal{B}$ as

$$\mathcal{B}(\mathsf{r}) := \mathsf{B} \star \mathsf{r}.$$

return $\mathcal{B}$.

4.1.2. Subspace correction method

The subspace correction (SC) method is a classical numerical method for solving linear equations. The SC smoother has more flexible parameters than the convolutional smoother in PDE-MgNet. For a linear system $\mathbf{Au} = \mathbf{f}$, a subspace correction $\mathcal{B}$ is determined by a subspace $\mathbb{G}$ which is usually represented by the range of a matrix $\mathbf{G}$, i.e.

$$\mathbb{G} = \mathrm{span}\{\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_L\} = \mathrm{range}(\mathbf{G}),$$

where $\mathbf{G} = (\mathbf{g}_1, ..., \mathbf{g}_L)$.

Notice that $\mathbf{g}_i$ has the same dimension as $\mathbf{f}$ and $\mathbf{u}$. Thus, we let Meta-NN export multi-channel tensors with the same shape as $\mathbf{f}$ and then reshape each channel to form $\mathbf{g}_i$. Details are given by **Algorithm 4**. The architecture of Meta-NN $\mathcal{G}$ in **Algorithm 4** needs a more careful design than the one in **Algorithm 3**. To select an appropriate subspace for traditional SC method is also difficult. The most popular choice is the Krylov subspace, i.e. $\mathbb{G}_{\mathrm{K}} = \{\mathbf{r}, \mathbf{Ar}, ..., \mathbf{A}^k\mathbf{r}\}$. If we write $\mathbb{G}_{\mathrm{K}} = \{\mathbf{f}_0(\mathbf{A})\mathbf{r}, \mathbf{f}_1(\mathbf{A})\mathbf{r}, ..., \mathbf{f}_k(\mathbf{A})\mathbf{r}\}$, and $\mathbf{f}_i(\mathbf{A}) = \mathbf{A}^i$. Inspired by such formulation, we design the Meta-NN in **Algorithm 4** as

Algorithm 4 $\mathcal{B} = B_{\mathrm{sc}}(\mathsf{r}, \mathsf{A}_\eta; \mathcal{G})$.

Hyper-parameters: Meta-NN $\mathcal{G}$
Inputs: r, A_η
Output: $\mathcal{B}$
1. Calculate subspace:

$$\mathsf{G} \leftarrow \mathcal{G}(\mathsf{r}, \mathsf{A}_\eta),$$

where G is a tensor with shape $L \times K \times J \times I$.
2. Reshape the tensor G to $L \times KJI$ matrix, and write its transpose as $\mathbf{G}$, which is a $KJI \times L$ matrix.
3. Do subspace correction with the subspace $\mathbb{G} = \mathrm{range}(\mathbf{G})$:

$$\mathbf{S}_\eta \leftarrow \mathbf{A}_\eta\mathbf{G}.$$

$$\mathbf{e} = \mathbf{G}(\mathbf{G}^\top\mathbf{S}_\eta)^{-1}\mathbf{G}^\top\mathbf{r}.$$

4. Define the effect of $\mathcal{B}$ as

$$\mathcal{B}(\mathsf{r}, \mathbf{A}_\eta; \mathcal{G}) := \mathrm{Reshape}(\mathbf{e}).$$

where $\mathrm{Reshape}(\cdot)$ means to reshape $\mathbf{e}$ to the same shape as tensor r.
return $\mathcal{B}$.

$$\mathcal{G}_\theta(\mathsf{r}, \mathsf{A}_\eta) = \mathcal{N}_{\mathrm{FC}_\theta(\mathsf{A}_\eta)}(\mathsf{r}). \tag{19}$$

Here, $\mathcal{N}_\gamma$ is a CNN used to convert r into a multi-channel tensor, and each channel of the output plays the role as a $\mathbf{f}_i(\mathbf{A})$ in $\mathbb{G}_{\mathrm{K}}$. The weights γ of $\mathcal{N}_\gamma$ are the output of the neural network FC_θ. In this paper, the CNN $\mathcal{N}_\gamma$ is a 3-layer Dense-Net block [26] and FC_θ is a 2-layer fully connected neural network. For the Dense-Net block, we use l to represent the channel number. An illustration of this Meta-NN is given in **Fig. 8**.

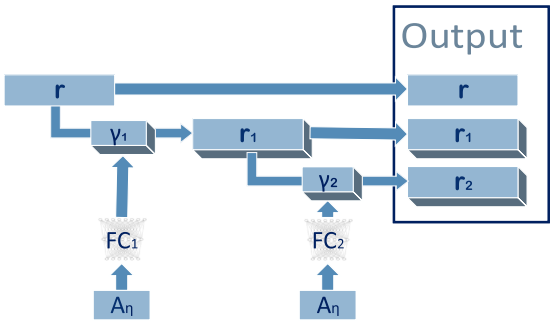

Fig. 8. The architecture of Meta-NN with 3 output channels as an example.

4.2. Training of Meta-MgNet

Suppose the distribution of η is Z and for each given η the distribution of f is F_η. Let the number of samples of η be M_{p}. For each η, we sample f for $M_{\text{m-train}}$ times and generate the data set $\mathfrak{X}_{F,\eta}$. Similar as the loss (17), we consider the following loss function

$$Loss = E_{\eta\sim Z,\mathsf{f}\sim F_\eta}\frac{||\mathsf{f}-\mathsf{A}_\eta\star\mathsf{u}_T||^2}{||\mathsf{f}||^2}\approx\frac{1}{M_{\mathrm{p}}M_{\text{m-train}}}\sum_{\eta\in\mathfrak{Z}}\sum_{\mathsf{f}\in\mathfrak{X}_{F,\eta}}\frac{||\mathsf{f}-\mathsf{A}_\eta\star\mathsf{u}_T||^2}{||\mathsf{f}||^2}.$$

In our experiments, we choose $T=1$ following [27].

Remark 1. We can fine-tune the trained model when we are given a PDE with a new η, just like what meta-learning usually does. However, as shown in **Appendix I**, it brings little benefit and thus we shall omit fine-tuning.

4.3. Convergence analysis

This section analyzes the convergence of the proposed Meta-MgNet (16) with SC for Poisson's equation. We assume that the discretization schemes are either FDM with 7-point (or 9-point) stencil or FEM with P1 or Q1 elements. Now suppose the approximation of $\mathbf{u}$ is $\mathbf{u}_t$, then the two-grid MG iteration can be written as

$$\begin{aligned}
\mathbf{r}_{t+\frac{1}{2}} &= \mathbf{f}-\mathbf{A}\mathbf{u}_t = \mathbf{A}(\mathbf{u}-\mathbf{u}_t)\\
\mathbf{u}_{t+\frac{1}{2}} &= \mathbf{u}_t + \mathbf{P}\mathbf{A}_c^{-1}\mathbf{P}^\top\mathbf{r}_{t+\frac{1}{2}}\\
\mathbf{r}_{t+1} &= \mathbf{f}-\mathbf{A}\mathbf{u}_{t+\frac{1}{2}} = \mathbf{A}(\mathbf{u}-\mathbf{u}_{t+\frac{1}{2}})\\
\\
\mathbf{u}_{t+1} &= \mathbf{u}_{t+\frac{1}{2}} + \mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{r}_{t+1}.
\end{aligned}$$

Then we have the recurrence relation of the error from t to $t+1$:

$$||\mathbf{u}-\mathbf{u}_{t+1}||_{\mathbf{A}} = ||(\mathbf{I}-\mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A})(\mathbf{u}-\mathbf{u}_{t+\frac{1}{2}})||_{\mathbf{A}} = ||(\mathbf{I}-\mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A})(\mathbf{I}-\mathbf{C})(\mathbf{u}-\mathbf{u}_t)||_{\mathbf{A}},$$

where $\mathbf{C}=\mathbf{P}\mathbf{A}_c^{-1}\mathbf{P}^\top\mathbf{A}$.

By [19], if the prolongation $\mathbf{P}$ is given by 7-point stencil or 9-point stencil, it is obviously that

$$||(\mathbf{I}-\mathbf{C})\mathbf{v}||_{\mathbf{A}}^2 \le ||\mathbf{v}||_{\mathbf{A}}^2,\ \forall\mathbf{v}\in\mathbb{V}_h, \tag{20}$$

and there is a constant $c_a>0$ s.t. $\forall\mathbf{v}\in\mathbb{V}_h$,

$$||(\mathbf{I}-\mathbf{C})\mathbf{v}||_{\mathbf{A}}^2 \le c_a{\rho_{\mathbf{A}}}^{-1}||\mathbf{v}||_{\mathbf{A}^2}^2, \tag{21}$$

where $\rho_{\mathbf{A}}$ is the spectral radius of $\mathbf{A}$.

Now, we introduce the following assumptions (see also [48,49]).

Assumption 4.1. *For a given* $\mathbf{r}$*, we assume that the associated* $\mathbf{B}_{\mathbf{r}}$ *satisfies:*

1. $\mathbf{B}_{\mathbf{r}}$ *is semi-symmetric positive defined (SSPD) and* $\mathbf{B}_{\mathbf{r}}\mathbf{A}\mathbf{B}_{\mathbf{r}}=\mathbf{B}_{\mathbf{r}}$,

2. *There is a constant $c_s > 0$ independent with $\mathbf{r}$, s.t.*

$$||\mathbf{r}||^2 \le c_s \rho_{\mathbf{A}} \mathbf{r}^\top \mathbf{B}_{\mathbf{r}} \mathbf{r}. \tag{22}$$

Then, we have the following convergence theorem for Meta-MgNet (see also [2]).

Theorem 2. *Let $\{\mathbf{u}_t\}$ be the sequence generated by the Meta-MgNet (16). Then, we have the convergence estimation*

$$||\mathbf{u} - \mathbf{u}_t||_{\mathbf{A}} \le \delta^{\frac{t}{2}} ||\mathbf{u} - \mathbf{u}_0||_{\mathbf{A}}, \quad \text{with } \delta = 1 - \frac{1}{c_a c_s}.$$

Proof. It suffices to show that

$$||\mathbf{u} - \mathbf{u}_{t+1}||_{\mathbf{A}} \le \delta^{\frac{1}{2}} ||\mathbf{u} - \mathbf{u}_t||_{\mathbf{A}}. \tag{23}$$

If $\mathbf{B}_{\mathbf{r}}$ satisfies **Assumption 4.1**, we have

$$\begin{aligned}
||\mathbf{u} - \mathbf{u}_{t+1}||_{\mathbf{A}}^2 &= ||(\mathbf{I} - \mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A})(\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}})||_{\mathbf{A}}^2 \\
&= (\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}})^\top (\mathbf{I} - \mathbf{A}\mathbf{B}_{\mathbf{r}_{t+1}})\mathbf{A}(\mathbf{I} - \mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A})(\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}) \quad (\text{by } \mathbf{B}_{\mathbf{r}_{t+1}} \text{ is SSPD}) \\
&= (\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}})^\top (\mathbf{A} - 2\mathbf{A}\mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A} + \mathbf{A}\mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A}\mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A})(\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}) \\
&= (\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}})^\top (\mathbf{A} - \mathbf{A}\mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A})(\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}) \quad (\text{by } \mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A}\mathbf{B}_{\mathbf{r}_{t+1}} = \mathbf{B}_{\mathbf{r}_{t+1}}) \\
&= ||\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}||_{\mathbf{A}}^2 - (\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}})^\top \mathbf{A}\mathbf{B}_{\mathbf{r}_{t+1}}\mathbf{A}(\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}) \\
&= ||\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}||_{\mathbf{A}}^2 - \mathbf{r}_{t+1}^\top \mathbf{B}_{\mathbf{r}_{t+1}} \mathbf{r}_{t+1} \\
&\le ||\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}||_{\mathbf{A}}^2 - c_s^{-1}\rho_{\mathbf{A}}^{-1} \mathbf{r}_{t+1}^\top \mathbf{r}_{t+1} \quad (\text{by } (22)) \\
&= ||\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}||_{\mathbf{A}}^2 - c_s^{-1}\rho_{\mathbf{A}}^{-1} (\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}})^\top \mathbf{A}^2 (\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}) \\
&= ||\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}}||_{\mathbf{A}}^2 - c_s^{-1}\rho_{\mathbf{A}}^{-1} ||(\mathbf{u} - \mathbf{u}_{t+\frac{1}{2}})||_{\mathbf{A}^2}^2
\end{aligned}$$

Write $\mathbf{w}_t = \mathbf{u} - \mathbf{u}_{t+\frac{1}{2}} = (\mathbf{I} - \mathbf{C})(\mathbf{u} - \mathbf{u}_t)$, with (20) and (21), we have

$$||\mathbf{u} - \mathbf{u}_{t+1}||_{\mathbf{A}}^2 \le ||\mathbf{w}||_{\mathbf{A}}^2 - c_s^{-1}\rho_{\mathbf{A}}^{-1} ||\mathbf{w}||_{\mathbf{A}^2} \le ||\mathbf{u} - \mathbf{u}_t||_{\mathbf{A}}^2 - c_s^{-1}c_a^{-1} ||\mathbf{u} - \mathbf{u}_t||_{\mathbf{A}}^2 = \delta ||\mathbf{u} - \mathbf{u}_t||_{\mathbf{A}}^2.$$

Thus, (23) is derived.

Now, we only need to verify that $\mathbf{G} = \mathbf{G}(\mathbf{r})$ given in **Algorithm 4** stratifies **Assumption 4.1**. Indeed, we have

$$\mathbf{B}_{\mathbf{r}} = \mathbf{G}(\mathbf{G}^\top \mathbf{A}\mathbf{G})^{-1}\mathbf{G}^\top.$$

For the first assumption,

$$\mathbf{A} \text{ is SPD} \Rightarrow \mathbf{G}^\top\mathbf{A}\mathbf{G} \text{ is SPD} \Rightarrow (\mathbf{G}^\top\mathbf{A}\mathbf{G})^{-1} \text{ is SPD} \Rightarrow \mathbf{B}_{\mathbf{r}} = \mathbf{G}(\mathbf{G}^\top\mathbf{A}\mathbf{G})^{-1}\mathbf{G}^\top \text{ is SSPD}.$$

And

$$\mathbf{B}_{\mathbf{r}}\mathbf{A}\mathbf{B}_{\mathbf{r}} = \mathbf{G}(\mathbf{G}^\top\mathbf{A}\mathbf{G})^{-1}\mathbf{G}^\top\mathbf{A}\mathbf{G}(\mathbf{G}^\top\mathbf{A}\mathbf{G})^{-1}\mathbf{G}^\top = \mathbf{G}(\mathbf{G}^\top\mathbf{A}\mathbf{G})^{-1}\mathbf{G}^\top = \mathbf{B}_{\mathbf{r}}.$$

For the second assumption, without loss of generality, we assume $||\mathbf{r}|| = 1$, then we have $||\mathbf{r}||_{\mathbf{A}}^2 \le \rho_{\mathbf{A}} ||\mathbf{r}||^2 = \rho_{\mathbf{A}}$. Write $\mathbf{G} = [\mathbf{g}_1, ..., \mathbf{g}_L]$, and $\mathbf{g}_i, i = 1, 2, ..., L$ satisfy $\mathbf{g}_i^\top \mathbf{A}\mathbf{g}_j = \delta_{ij}$. Choose $\mathbf{g}_1 = \dfrac{\mathbf{r}}{||\mathbf{r}||_{\mathbf{A}}}$. Then, we obtain

$$\mathbf{r}^\top \mathbf{B}_{\mathbf{r}} \mathbf{r} \ge (\mathbf{r}^\top \mathbf{g}_1)^2 = \frac{1}{||\mathbf{r}||_{\mathbf{A}}^2} \ge \frac{1}{\rho_{\mathbf{A}}}.$$

This means that (22) is satisfied if $\mathbf{r} \in \text{range}(\mathbf{G})$, which is obvious from the design of the Meta-NN for B_{sc}. □

5. Numerical experiments

In this section, we evaluate the performance of Meta-MgNet through a series of numerical experiments. In **Section 5.1** and **5.2**, we apply Meta-MgNet to 2D and 3D anisotropic diffusion equations on domain $\Omega = [0,1]^d, d = 2, 3$. In **Section 5.3**, we demonstrate why it is challenging to train a PDE-MgNet that generalizes well for all η. In **Section 5.4**, we include more practical examples.

In the experiments, we compare Meta-MgNet with PDE-MgNet and MG method. Both Meta-MgNet and PDE-MgNet are trained on a data set created by a set of η, which will be later described in detail. Furthermore, we also train PDE-MgNet for each individual η and denote the trained model as PDE-MgNet-η. During testing, we only apply PDE-MgNet-η on the test data generated by the same η as during training. Thus, PDE-MgNet-η demonstrates the best accuracy of PDE-MgNet may achieve while it is impractical since it requires retraining of PDE-MgNet for each individual η. For the classical MG method, we choose the Krylov subspace smoother, GS smoother, line-Jacobi smoother, and damped Jacobi smoother. Since the GS and line-Jacobi smoother are challenging to implement efficiently with GPU, we use Matlab on CPU instead. All other algorithms are implemented in PyTorch on GPU.

5.1. 2D anisotropic diffusion equations

We consider the anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \tag{24}$$

where $C = C(\epsilon,\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$ is a 2×2 matrix, $\epsilon < 1, \theta \in [0,\pi]$.

5.1.1. Settings: training, testing and hyperparameter selection

(1) For the training set, we randomly sample $M_{\text{p}} = 20$ different sets of parameters η from an interval I_{train}. The interval I_{train} is different for different PDEs. Thus, we will specify it later for each experiment. For each given η, we randomly sample $M_{\text{m-train}} = 100$ right-hand-side function, and each entry of f is sampled from the Gaussian distribution $N(0,1)$. We use ADAM method and the unsupervised learning loss (17) to train PDE-MgNet for 50 epochs and Meta-MgNet for 20 epochs. Since no matter if Meta-MgNet is fine-tuned, the result is almost the same, we skip the fine-tuning stage of Meta-MgNet. (See **Appendix I** for an ablation study on fine-tuning.) The learning rate for ADAM is 0.02 and the batch size is 64.

(2) For the test set, we choose some specific η from a set I_{test}, and for each η we randomly sample $M_{\text{m-test}} = 10$ right-hand-side function. The stopping criterion for all compared algorithms is chosen as

$$\frac{||\mathbf{f} - \mathbf{A}_\eta \mathbf{u}_t||_2}{||\mathbf{f}||_2} < 10^{-6}.$$

Number of iterations and wall time are used as metrics to compare the performance of different algorithms. Furthermore, we use "mean±std" to show the average and standard deviation of the number of iteration and wall time over the $M_{\text{m-test}}$ samples.

(3) We use $N \times N$ rectangular mesh and Q_1-element to discretize the PDEs. We select $N = 256$ and the number of layers $J = 5$. The prolongations $\mathcal{P}$ and restrictions $\mathcal{R}$ are given by the traditional 9-point stencil, i.e. the kernel of $\mathcal{P}$ and $\mathcal{R}$ is

$$\mathsf{P} = \mathsf{R} = \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix}.$$

The $\mathcal{A}^\ell$ on coarse grid is also given by the geometric multigrid method, and it is easy to verify that each A^ℓ is equal for Q_1-element. We chose \-Cycle structure for all algorithms with $\nu_1 = 2, \nu_2 = \cdots = \nu_J = 1$. (We tried several different settings of ν_l and found that $\nu_1 = 2, \nu_2 = \cdots = \nu_J = 1$ is the best option. See **Appendix II** for the corresponding ablation study.)

(4) For PDE-MgNet, the kernel size of smoother at each layer is 7×7. For Meta-MgNet, we choose SC method B_{sc} in **Algorithm 4** and use the Meta-NN in (19). The kernel size of the output is 7×7 and the increase channel number of Meta-NN is $l = 3$. The size of the hidden layer of the fully connected neural network FC_θ is 200. The number of the parameters of one single Meta-NN is 122.2K. The Meta-MgNet in this numerical experiment contains 5 Meta-NNs, which means the total number of the parameters of Meta-MgNet is around 611K. Note that a comparison between B_{d} and B_{sc} is given in **Appendix III**, it shows that B_{sc} is the better option. Thus, we only present the numerical result of B_{sc} in the main body of this paper.

Table 2
The number of iterations (when the stopping criteria is met) and the wall time of Meta-MgNet, PDE-MgNet and MG method. "-" means the algorithm does not converge within 10^4 iterations.

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η	MG (Krylov)	MG (GS)	MG (line-Jacobi)	MG (Jacobi)
$\epsilon = 1$	4.0 ± 0.00	–	7.0 ± 0.00	4.0 ± 0.00	10.0 ± 0.00	14.0 ± 0.00	15.0 ± 0.00
$\epsilon = 10^{-1}$	7.5 ± 0.50	19.2 ± 0.40	21.2 ± 0.60	7.9 ± 0.30	33.7 ± 0.48	13.0 ± 0.00	90.2 ± 0.98
$\epsilon = 10^{-2}$	35.1 ± 1.04	178.9 ± 2.74	149.7 ± 3.44	52.5 ± 0.81	253.6 ± 4.19	13.0 ± 0.00	752.8 ± 12.23
$\epsilon = 10^{-3}$	171.6 ± 6.34	$1.2e3 \pm 12.85$	910.9 ± 15.64	345.9 ± 3.88	$1.9e3 \pm 25.56$	13.0 ± 0.00	$5.6e3 \pm 119.42$
$\epsilon = 10^{-4}$	375.2 ± 5.88	–	$3.1e3 \pm 35.70$	$2.2e3 \pm 27.94$	–	11.0 ± 0.00	–
$\epsilon = 10^{-5}$	797.8 ± 12.76	–	$9.9e3 \pm 40.81$	$7.6e3 \pm 81.96$	–	11.0 ± 0.00	–
wall time							
$\epsilon = 1$	0.03 ± 0.00	–	$\mathbf{0.02 \pm 0.00}$	$\mathbf{0.02 \pm 0.00}$	0.14 ± 0.01	0.34 ± 0.01	0.04 ± 0.00
$\epsilon = 10^{-1}$	0.05 ± 0.00	0.05 ± 0.00	0.06 ± 0.00	$\mathbf{0.04 \pm 0.00}$	0.48 ± 0.02	0.32 ± 0.01	0.23 ± 0.00
$\epsilon = 10^{-2}$	$\mathbf{0.22 \pm 0.01}$	0.44 ± 0.01	0.37 ± 0.01	0.25 ± 0.00	3.47 ± 0.15	0.32 ± 0.01	1.85 ± 0.03
$\epsilon = 10^{-3}$	1.06 ± 0.04	3.04 ± 0.03	2.28 ± 0.05	1.64 ± 0.02	27.33 ± 0.68	$\mathbf{0.32 \pm 0.01}$	13.95 ± 0.29
$\epsilon = 10^{-4}$	2.31 ± 0.03	–	7.69 ± 0.13	10.56 ± 0.14	–	$\mathbf{0.27 \pm 0.01}$	–
$\epsilon = 10^{-5}$	4.91 ± 0.08	–	24.49 ± 0.14	35.67 ± 0.40	–	$\mathbf{0.27 \pm 0.02}$	–

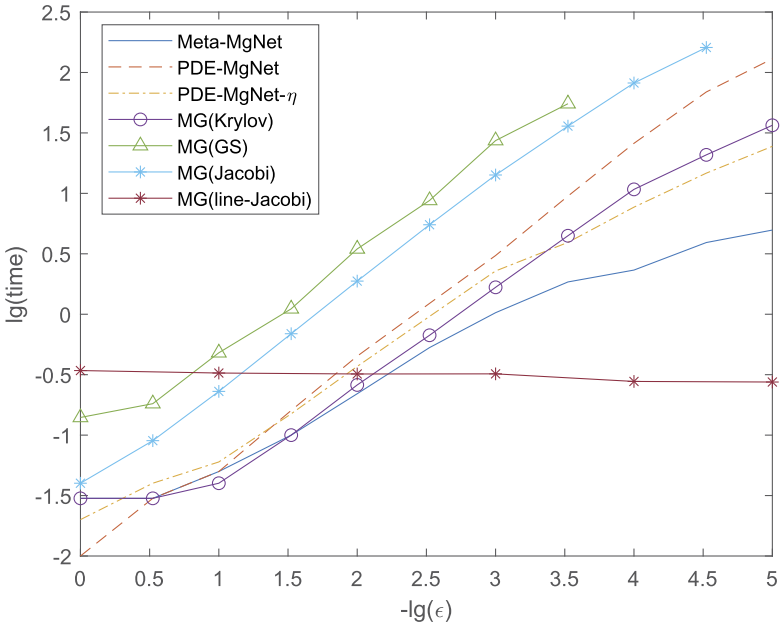

Fig. 9. Wall time of Meta-MgNet, PDE-MgNet, and the MG methods while $\theta = 0$ and ϵ varies.

We compare number of iterations and wall time of Meta-MgNet, PDE-MgNet and MG method. We consider two different generalization scenarios that will be called "in-distribution generalization" and "out-of-distribution (OoD) transfer". For in-distribution generalization, we have $I_{\text{test}} \subset I_{\text{train}}$, while for OoD transfer, the parameter η of the test data is entirely outside the interval of the training data, i.e. $I_{\text{test}} \subset I^c_{\text{train}}$.

5.1.2. In-distribution generalization

In this group of experiments, the training set of PDE-MgNet and Meta-MgNet is generated by fixing $\theta = 0$ in **Table 2** and $\theta = 0.1\pi$ in **Table 3** and randomly sampling ϵ with distribution $\lg \frac{1}{\epsilon} \sim U[0, 5]$. The **Table 2**, **Table 3** and **Fig. 9** show Meta-MgNet has overall better performance than PDE-MgNet and MG methods, while the advantage is significant when ϵ is small. It is worth mentioning that the line-Jacobi smoother can only be applied to several specific θ, such as $0, \frac{\pi}{4}, \frac{\pi}{2}$, thus line-Jacobi smoother is limited in practical applications.

5.1.3. Out-of-distribution (OoD) transfer

The first group of experiments are with the fixed $\theta = 0$. The training set is generated with $\lg \frac{1}{\epsilon} \sim U[2, 3]$ and the test set is generated with $\epsilon = 1, 10^{-1}, 10^{-4}, 10^{-5}$ (i.e. a test for OoD transfer with respect to ϵ). For the second group of

Table 3
The number of iterations (when the stopping criteria is met) and the wall time of Meta-MgNet, PDE-MgNet and MG method. "-" means the algorithm does not converge within 10^4 iterations.

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η	MG (Krylov)	MG (GS)	MG (line-Jacobi)	MG (Jacobi)
$\epsilon = 1$	4.0 ± 0.00	–	17.00 ± 0.00	4.0 ± 0.00	10.0 ± 0.00	–	15.0 ± 0.00
$\epsilon = 10^{-1}$	5.4 ± 0.49	136.7 ± 1.95	68.30 ± 2.69	8.0 ± 0.30	27.3 ± 0.48	–	64.70 ± 0.64
$\epsilon = 10^{-2}$	28.5 ± 0.50	$1.0e3 \pm 25.39$	861.40 ± 24.53	45.4 ± 0.49	187.0 ± 2.53	–	476.30 ± 4.78
$\epsilon = 10^{-3}$	94.2 ± 1.33	$3.8e3 \pm 183.22$	$1.9e3 \pm 60.41$	142.8 ± 1.89	707.6 ± 15.24	–	$1.8e3 \pm 44.47$
$\epsilon = 10^{-4}$	129.4 ± 2.42	$5.3e3 \pm 180.33$	$3.8e3 \pm 94.79$	187.8 ± 3.79	990.2 ± 19.03	–	$2.5e3 \pm 62.01$
$\epsilon = 10^{-5}$	134.8 ± 2.86	$5.6e3 \pm 168.51$	$4.1e3 \pm 110.96$	195.9 ± 2.43	$1.0e3 \pm 37.45$	–	$2.6e3 \pm 79.24$
wall time							
$\epsilon = 1$	0.04 ± 0.00	–	0.05 ± 0.00	$\mathbf{0.03 \pm 0.00}$	0.14 ± 0.01	–	0.04 ± 0.00
$\epsilon = 10^{-1}$	$\mathbf{0.05 \pm 0.00}$	0.36 ± 0.01	0.18 ± 0.01	$\mathbf{0.05 \pm 0.00}$	0.29 ± 0.02	–	0.17 ± 0.00
$\epsilon = 10^{-2}$	$\mathbf{0.21 \pm 0.00}$	2.64 ± 0.06	2.23 ± 0.06	0.24 ± 0.00	2.05 ± 0.08	–	1.26 ± 0.01
$\epsilon = 10^{-3}$	$\mathbf{0.69 \pm 0.01}$	9.89 ± 0.53	4.90 ± 0.17	0.75 ± 0.01	7.78 ± 0.21	–	4.79 ± 0.11
$\epsilon = 10^{-4}$	$\mathbf{0.94 \pm 0.02}$	13.87 ± 0.48	9.90 ± 0.24	0.98 ± 0.02	10.76 ± 0.25	–	6.66 ± 0.15
$\epsilon = 10^{-5}$	$\mathbf{0.98 \pm 0.02}$	14.56 ± 0.44	10.69 ± 0.30	1.02 ± 0.01	11.36 ± 0.51	–	7.03 ± 0.21

Table 4
The mean and std of the number of iterations (when the stopping criteria is met) and the wall time of Meta-MgNet, PDE-MgNet and MG method on the testing set. "-" means the algorithm does not converge within 10^4 iterations.

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η	MG (Krylov)	MG (GS)	MG (line-Jacobi)	MG (Jacobi)
$\epsilon = 1$	7.0 ± 0.00	–	7.0 ± 0.00	4.0 ± 0.00	10.0 ± 0.00	14.0 ± 0.00	15.0 ± 0.00
$\epsilon = 10^{-1}$	10.0 ± 0.00	23.0 ± 0.00	21.2 ± 0.60	7.9 ± 0.30	33.7 ± 0.48	13.0 ± 0.00	90.2 ± 0.98
$\epsilon = 10^{-4}$	340.7 ± 3.52	$5.8e3 \pm 121.90$	$3.1e3 \pm 35.70$	$2.2e3 \pm 27.94$	–	11.0 ± 0.00	–
$\epsilon = 10^{-5}$	817.2 ± 97.97	–	$9.9e3 \pm 40.81$	$7.6e3 \pm 81.96$	–	11.0 ± 0.00	–
wall time							
$\epsilon = 1$	0.05 ± 0.00	–	$\mathbf{0.02 \pm 0.00}$	$\mathbf{0.02 \pm 0.00}$	0.14 ± 0.01	0.24 ± 0.01	0.04 ± 0.00
$\epsilon = 10^{-1}$	0.07 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{0.04 \pm 0.00}$	0.48 ± 0.02	0.32 ± 0.01	0.23 ± 0.00
$\epsilon = 10^{-4}$	2.08 ± 0.02	14.38 ± 0.32	7.69 ± 0.13	10.56 ± 0.14	–	$\mathbf{0.27 \pm 0.01}$	–
$\epsilon = 10^{-5}$	4.99 ± 0.59	–	24.49 ± 0.14	35.67 ± 0.40	–	$\mathbf{0.27 \pm 0.02}$	–

experiments, the training set is generated with $\theta \sim U[0.125\pi, 0.375\pi]$ and $\lg \frac{1}{\epsilon} \sim U[0, 5]$, while the test set is generated with $\theta = 0.05\pi, 0.12\pi, 0.4\pi$ and 0.5π (i.e. a test for OoD transfer with respect to θ).

Table 4 shows that Meta-MgNet has superior ability of OoD transfer with respect to ϵ in comparison with PDE-MgNet, while **Table 5** shows that Meta-MgNet is noticeably superior in OoD transfer with respect to θ in comparison with PDE-MgNet. Note that classical methods are not learning-based, and hence they do not have the issue of OoD transfer. Thus, the results of MG methods in Table 4 are copied from Table 2.

5.1.4. Further discussions

It is also worth noting that, for both the scenarios of in-distribution generalization and OoD transfer, Meta-MgNet even outperforms PDE-MgNet-η which uses the exact same η for both training and testing. This shows the benefit of treating the problem of solving parameterized PDEs as a multi-task learning problem. From what it seems, the hypernetwork, i.e. Meta-NN, is able to extract certain common structure hidden within the tasks which helps with solving each individual task.

The training time of Meta-MgNet is around 45 minutes, while it is around 10 minutes for PDE-MgNet-η for each η. Although training Meta-MgNet takes more times than PDE-MgNet-η for each η. In practical application, especially multi-query scenarios, the utility of PDE-MgNet-η can be significantly reduced due to the constant retraining. Therefore, Meta-MgNet has a clear overall advantage.

5.2. 3D anisotropic diffusion equations

Consider the following 3D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \quad \text{and} \quad C = \begin{pmatrix} \epsilon_0 & & \\ & \epsilon_1 & \\ & & \epsilon_2 \end{pmatrix},$$

Table 5
The mean and std of the number of iterations (when the stopping criteria is met) and the wall time of Meta-MgNet, PDE-MgNet and MG method on the testing set. "-" means the algorithm does not converge within 10^4 iterations.

#iterations	Meta-MgNet, $\theta = 0.05\pi$	PDE-MgNet, $\theta = 0.05\pi$	Meta-MgNet, $\theta = 0.12\pi$	PDE-MgNet, $\theta = 0.12\pi$
$\epsilon = 1$	3.0 ± 0.00	–	3.0 ± 0.00	–
$\epsilon = 10^{-1}$	10.6 ± 0.49	–	10.1 ± 0.30	132.6 ± 3.10
$\epsilon = 10^{-2}$	71.5 ± 1.57	–	72.0 ± 1.61	566.3 ± 11.36
$\epsilon = 10^{-3}$	322.4 ± 7.03	–	233.2 ± 7.49	$2.1e3 \pm 55.09$
$\epsilon = 10^{-4}$	526.7 ± 14.64	–	306.0 ± 7.78	$2.8e3 \pm 150.40$
$\epsilon = 10^{-5}$	557.4 ± 14.72	–	314.0 ± 4.86	$2.8e3 \pm 33.31$
wall time				
$\epsilon = 1$	0.03 ± 0.00	–	0.03 ± 0.00	–
$\epsilon = 10^{-1}$	0.07 ± 0.00	–	0.07 ± 0.00	0.35 ± 0.01
$\epsilon = 10^{-2}$	0.46 ± 0.01	–	0.46 ± 0.01	1.49 ± 0.04
$\epsilon = 10^{-3}$	2.05 ± 0.04	–	1.48 ± 0.05	5.42 ± 0.18
$\epsilon = 10^{-4}$	3.34 ± 0.09	–	1.95 ± 0.05	7.49 ± 0.45
$\epsilon = 10^{-5}$	3.54 ± 0.09	–	1.99 ± 0.03	7.49 ± 0.15
#iterations	Meta-MgNet, $\theta = 0.4\pi$	PDE-MgNet, $\theta = 0.4\pi$	Meta-MgNet, $\theta = 0.5\pi$	PDE-MgNet, $\theta = 0.5\pi$
$\epsilon = 1$	3.0 ± 0.00	–	3.0 ± 0.0	–
$\epsilon = 10^{-1}$	9.0 ± 0.00	51.7 ± 1.27	8.9 ± 0.30	49.3 ± 1.42
$\epsilon = 10^{-2}$	65.2 ± 1.54	434.5 ± 7.75	53.5 ± 1.12	428.8 ± 12.83
$\epsilon = 10^{-3}$	240.3 ± 3.41	$1.7e3 \pm 50.92$	262.9 ± 5.96	$2.8e3 \pm 16.81$
$\epsilon = 10^{-4}$	327.5 ± 5.33	$2.4e3 \pm 67.84$	526.4 ± 25.51	–
$\epsilon = 10^{-5}$	339.5 ± 6.92	$2.5e3 \pm 85.99$	908.7 ± 27.43	–
wall time				
$\epsilon = 1$	0.03 ± 0.00	–	0.03 ± 0.00	–
$\epsilon = 10^{-1}$	0.06 ± 0.00	0.14 ± 0.00	0.06 ± 0.00	0.13 ± 0.01
$\epsilon = 10^{-2}$	0.42 ± 0.01	1.15 ± 0.03	0.35 ± 0.01	1.13 ± 0.03
$\epsilon = 10^{-3}$	1.53 ± 0.02	4.48 ± 0.12	1.67 ± 0.04	7.34 ± 0.11
$\epsilon = 10^{-4}$	2.09 ± 0.03	6.38 ± 0.22	3.35 ± 0.16	–
$\epsilon = 10^{-5}$	2.15 ± 0.04	6.61 ± 0.31	5.76 ± 0.18	–

with $\Omega = [0,1]^3$ and $\epsilon_0, \epsilon_1, \epsilon_2 > 0$. Without loss of generality, we set $\epsilon_0 = 1$.

5.2.1. Settings: training, testing and hyperparameter selection

(1) The settings of training and testing are the same as the 2D case.
(2) We use $N \times N \times N$ rectangular mesh and use Q_1-element to discretize the PDE. Let $N = 64$ and the number of layers $J = 4$. The prolongations $\mathcal{P}$ and restrictions $\mathcal{R}$ are given by the traditional 9-point stencil. The $\mathcal{A}^\ell$ on the coarse grid is given by the geometry MG method. It is easy to verify that each A^ℓ is equal for Q_1-element. We chose \-Cycle structure of all algorithms with $\nu_1 = 2, \nu_2 = \cdots = \nu_J = 1$.
(3) For PDE-MgNet, the size of the kernel for the convolution smoother is $3 \times 3 \times 3$. For Meta-MgNet, we choose the SC method B_{sc} in **Algorithm 4** and use the Meta-NN in (19). We simply set the convolution smoother to be one layer CNN without activation, and the size of the kernel for the output is $7 \times 7 \times 7$. We set the number of channels of Meta-NN to $l = 3$.

5.2.2. In-distribution generalization

In this group of experiments, the training data set is generated by sampling ϵ_1 and ϵ_2 from distribution $\lg \frac{1}{\epsilon_1} \sim U[0,5]$ and $\lg \frac{1}{\epsilon_2} \sim U[0,5]$ respectively. **Table 6** shows that Meta-MgNet is more efficient than classic MG methods.

5.2.3. Out-of-distribution (OoD) transfer

In this group of experiments, the training data set is generated by sampling ϵ_1 and ϵ_2 from distribution $\lg \frac{1}{\epsilon_1} \sim U[3,4]$ and $\lg \frac{1}{\epsilon_2} \sim U[3,4]$ respectively. **Table 7** shows that Meta-MgNet has an overall best performance, and PDE-MgNet has trouble in OoD transfer with respect to ϵ_1 and ϵ_2.

Table 6
The mean and std of the number of iterations (when the stopping criteria is met) and the wall time of Meta-MgNet, PDE-MgNet and MG method on the testing set. "-" means the algorithm does not converge within 10^4 iterations.

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η	MG (Krylov)	MG (GS)	MG (Jacobi)
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-1})$	5.0 ± 0.00	11.0 ± 0.00	11.0 ± 0.00	7.0 ± 0.00	53.0 ± 1.05	–
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-2})$	13.0 ± 0.00	91.4 ± 1.20	46.6 ± 1.69	38.3 ± 1.00	159.9 ± 5.27	–
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-5})$	156.2 ± 3.57	$3.2e3\pm73.59$	475.0 ± 8.00	606.8 ± 30.08	–	–
$(\epsilon_1,\epsilon_2)=(10^{-2},10^{-2})$	10.3 ± 0.46	116.4 ± 0.49	54.3 ± 0.78	45.7 ± 0.78	178.4 ± 2.54	291.52 ± 9.14
$(\epsilon_1,\epsilon_2)=(10^{-2},10^{-5})$	73.4 ± 0.80	$3.5e3\pm77.82$	771.5 ± 9.39	631.70 ± 16.64	–	–
$(\epsilon_1,\epsilon_2)=(10^{-5},10^{-5})$	111.6 ± 0.92	$8.3e3\pm65.18$	$5.5e3\pm99.85$	$1.8e3\pm20.23$	–	–
wall time						
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-1})$	0.13 ± 0.00	0.09 ± 0.00	0.09 ± 0.00	$\mathbf{0.07\pm0.00}$	4.29 ± 0.10 -	–
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-2})$	$\mathbf{0.30\pm0.00}$	0.65 ± 0.01	0.34 ± 0.02	0.32 ± 0.01	14.2 ± 0.64	–
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-5})$	3.40 ± 0.08	22.53 ± 0.51	$\mathbf{3.34\pm0.06}$	4.93 ± 0.25	–	–
$(\epsilon_1,\epsilon_2)=(10^{-2},10^{-2})$	$\mathbf{0.25\pm0.01}$	0.82 ± 0.00	0.39 ± 0.01	0.38 ± 0.01	14.67 ± 0.61	6.00 ± 0.00
$(\epsilon_1,\epsilon_2)=(10^{-2},10^{-5})$	$\mathbf{1.62\pm0.02}$	24.73 ± 0.55	5.42 ± 0.06	5.12 ± 0.13	–	–
$(\epsilon_1,\epsilon_2)=(10^{-5},10^{-5})$	$\mathbf{2.45\pm0.02}$	58.87 ± 0.47	38.93 ± 0.71	14.90 ± 0.16	–	–

Table 7
The mean and std of the number of iterations (when the stopping criteria is met) and the wall time of Meta-MgNet, PDE-MgNet and MG method on the testing set. "-" means the algorithm does not converge within 10^4 iterations.

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η	MG (Krylov)	MG (GS)	MG (Jacobi)
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-1})$	10.0 ± 0.00	–	11.0 ± 0.00	7.0 ± 0.00	53.0 ± 1.05	–
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-2})$	43.1 ± 0.54	–	46.6 ± 1.69	38.3 ± 1.00	159.9 ± 5.27	–
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-5})$	755.4 ± 55.88	–	475.0 ± 8.00	606.8 ± 30.08	–	–
$(\epsilon_1,\epsilon_2)=(10^{-2},10^{-2})$	11.0 ± 0.00	–	54.3 ± 0.78	45.7 ± 0.78	178.4 ± 2.54	291.52 ± 9.14
$(\epsilon_1,\epsilon_2)=(10^{-2},10^{-5})$	106.7 ± 1.35	–	771.5 ± 9.39	631.7 ± 16.64	–	–
$(\epsilon_1,\epsilon_2)=(10^{-5},10^{-5})$	125.7 ± 2.19	–	$5.5e3\pm99.85$	$1.8e3\pm20.23$	–	–
wall time						
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-1})$	0.24 ± 0.01	–	0.09 ± 0.00	$\mathbf{0.07\pm0.00}$	4.29 ± 0.10 -	–
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-2})$	0.95 ± 0.01	–	0.34 ± 0.02	$\mathbf{0.32\pm0.01}$	14.2 ± 0.64	–
$(\epsilon_1,\epsilon_2)=(10^{-1},10^{-5})$	16.37 ± 1.22	–	$\mathbf{3.34\pm0.06}$	4.93 ± 0.25	–	–
$(\epsilon_1,\epsilon_2)=(10^{-2},10^{-2})$	$\mathbf{0.26\pm0.00}$	–	0.39 ± 0.01	0.38 ± 0.01	14.67 ± 0.61	6.00 ± 0.00
$(\epsilon_1,\epsilon_2)=(10^{-2},10^{-5})$	$\mathbf{2.34\pm0.03}$	–	5.42 ± 0.06	5.12 ± 0.13	–	–
$(\epsilon_1,\epsilon_2)=(10^{-5},10^{-5})$	$\mathbf{2.76\pm0.05}$	–	38.93 ± 0.71	14.90 ± 0.16	–	–

5.3. Why is it challenging to train a convergent PDE-MgNet for all η?

For both the 2D and 3D anisotropic diffusion equations, we found it challenging to train appropriate weights for PDE-MgNet that can generalize beyond its training setting. This is, in fact, the motivation of viewing solving parameterized PDEs as multi-task learning and introducing Meta-MgNet. In this subsection, we conduct a simple experiment to demonstrate this issue with PDE-MgNet.

Consider the 3D anisotropic diffusion equation. We choose two different distributions for the parameters D_1: $\lg_2\epsilon_1\sim U[-2,-1], \lg_2\epsilon_2\sim U[1,2]$ and D_2: $\lg_2\epsilon_1\sim U[1,2], \lg_2\epsilon_2\sim U[-2,-1]$ to train two PDE-MgNet. We present the weights of the convolution smoothers at the finest level ($\ell=1$), namely K^1 in (25) and K^2 in (26):

$$\mathsf{K}^1=\left[\begin{bmatrix}-0.0170 & -0.0503 & -0.0169\\ 0.0051 & -0.2370 & 0.0051\\ -0.0171 & -0.0504 & -0.0170\end{bmatrix}\begin{bmatrix}0.0389 & 0.2431 & 0.0389\\ -0.0696 & 1.1127 & -0.0695\\ 0.0388 & 0.2431 & 0.0389\end{bmatrix}\begin{bmatrix}-0.0171 & -0.0503 & -0.0170\\ 0.0050 & -0.2370 & 0.0051\\ -0.0170 & -0.0503 & -0.0169\end{bmatrix}\right], \tag{25}$$

$$\mathsf{K}^2=\left[\begin{bmatrix}-0.0127 & -0.0331 & -0.0126\\ 0.0339 & 0.2175 & 0.0340\\ -0.0126 & -0.0332 & -0.0127\end{bmatrix}\begin{bmatrix}0.0113 & -0.2041 & 0.0114\\ -0.0766 & 1.0647 & -0.0765\\ 0.0113 & -0.2041 & 0.0113\end{bmatrix}\times\begin{bmatrix}-0.0126 & -0.0332 & -0.0126\\ 0.0340 & 0.2175 & 0.0340\\ -0.0127 & -0.0331 & -0.0127\end{bmatrix}\right]. \tag{26}$$

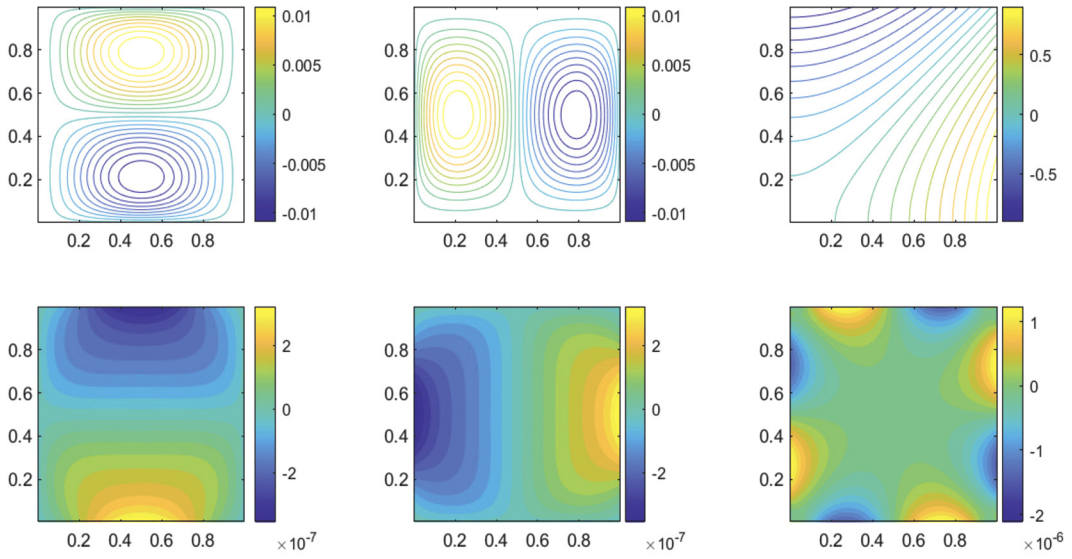

Fig. 10. The three pictures on the top row are the numerical solutions and the three at the bottom row are the error of u, v and p with $(a_x, a_y) = (0, 0)^\top$. The number of iterations is 367 and the wall time is 17.57 s for Meta-MgNet.

Noting the red numbers in K^1 and K^2, we can see that $\mathsf{K}^1_{-1,0,0}, \mathsf{K}^1_{1,0,0} < 0$ while $\mathsf{K}^2_{-1,0,0}, \mathsf{K}^2_{1,0,0} > 0$; and $\mathsf{K}^1_{0,-1,0}, \mathsf{K}^1_{0,1,0} > 0$ while $\mathsf{K}^2_{0,-1,0}, \mathsf{K}^2_{0,1,0} < 0$. Furthermore, if we use K^1 to smooth the PDEs with parameters generated from D_2, the error will increase, which means the convolution smoother with kernel K^1 is not fit for D_2. We have the same issue with K_2 and D_1. This phenomenon indicates that different distribution of parameters of the parameterized PDE may lead to weights of PDE-MgNet of contradictory effects. This is not only for PDE-MgNet, but rather an issue often occurs for supervised learning models. In contrast, Meta-MgNet handles this issue gracefully by adjusting the weights according to η in an adaptive fashion.

5.4. Ossen equations

The right-hand-side function $\underset{\sim}{f}$ in previous numerical experiments is randomly generated from the normal distribution. In this section, we include practical numerical examples to show the performance of Meta-MgNet. We adopt Ossen equations as an example:

$$\begin{cases} -\mu\Delta\underset{\sim}{u} + (\underset{\sim}{a}\cdot\nabla)\underset{\sim}{u} + \nabla p = \underset{\sim}{f}, & \text{in } \Omega, \\ -\mathrm{div}\underset{\sim}{u} = \underset{\sim}{0}, & \text{in } \Omega, \\ \underset{\sim}{u} = \underset{\sim}{0}, & \text{on } \partial\Omega, \end{cases}$$

where $\underset{\sim}{u} = (u, v)^\top$, $\mu = \dfrac{1}{Re}$, Re is the Reynold number, and $\underset{\sim}{a} = (a_x, a_y)^\top$. Without loss of generality, we let $\mu = 1$. We choose $\underset{\sim}{u} = \begin{pmatrix} -2xy^2(1-x)(1-2x)(1-y)^2 \\ 2x^2y(1-y)(1-2y)(1-x)^2 \end{pmatrix}$ and $p = x^2 - y^2$ as the solution and calculate the analytic form of the corresponding right-hand-side function $\underset{\sim}{f}$. The training data set is construct from sampling $a_x \sim U[0, 200]$ and $a_y \sim U[0, 200]$. We use the MAC scheme [38,6] to discretize Ossen equations and the mesh size is 512×512. Except for the settings mentioned above, other settings are the same as **Section 5.1**. Numerical solutions and error maps for a few different $\underset{\sim}{a}$ are presented in **Fig. 10–13** with the number of iterations and the wall time of Meta-MgNet recorded in the captions.

6. Conclusions and future work

Solving parameterized PDEs is an essential and yet challenging task. In this paper, we provided a new perspective on the problem by viewing it as multi-task learning. With this, we proposed a new meta-learning based solver called Meta-MgNet by introducing a carefully designed hypernetwork (called Meta-NN) to the PDE-MgNet. Numerical experiments on 2D and 3D anisotropic diffusion equations showed that Meta-MgNet significantly outperforms the supervised learning-based PDE-MgNet and classical MG methods. Furthermore, Meta-MgNet manifested a clear advantage in training and generalization over PDE-MgNet, which demonstrated the feasibility of the proposed multi-task perspective and meta-learning approach to solving parameterized PDEs.

This paper only discussed using meta-learning to improve smoothers because the prolongations and restrictions in classic MG methods are efficient enough to solve the three PDEs considered in this paper. As for some other PDEs such as Helmholtz

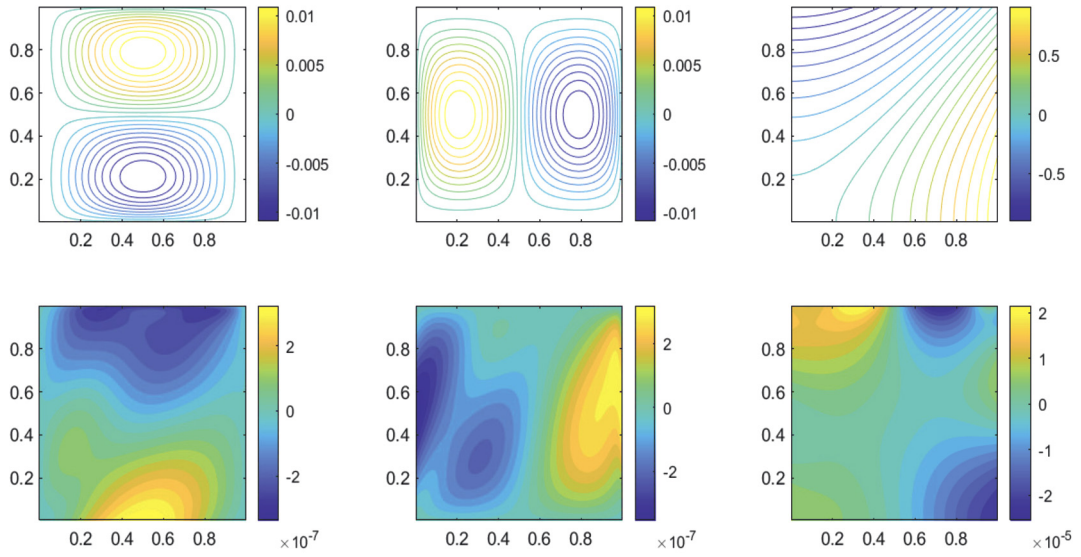

Fig. 11. The three pictures on the top row are the numerical solutions and the three at the bottom row are the error of u, v and p with $(a_x, a_y) = (50, 100)^\top$. The number of iterations is 281 and the wall time is 13.6 s for Meta-MgNet.

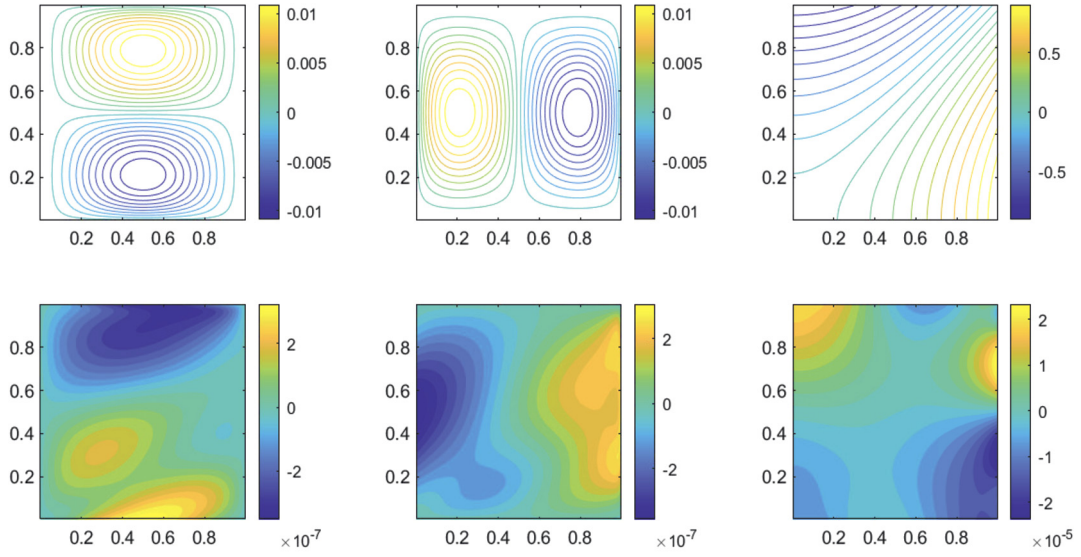

Fig. 12. The three pictures on the top row are the numerical solutions and the three at the bottom row are the error of u, v and p with $(a_x, a_y) = (100, 50)^\top$. The number of iterations is 275 and the wall time is 13.38 s for Meta-MgNet.

equations, the convolutional prolongations and restrictions may not work well. Therefore, it is worth exploring a data-driven approach to improve prolongations and restrictions as well. Furthermore, we only considered uniform mesh in this paper. We may consider generalizing MgNet or Meta-MgNet to nonuniform meshes, such as the triangular mesh, by exploiting tools from geometric deep learning, such as graph (convolutional) neural networks.

CRediT authorship contribution statement

Yuyan Chen: Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft. **Bin Dong:** Conceptualization, Funding acquisition, Methodology, Resources, Supervision, Writing – review & editing. **Jinchao Xu:** Funding acquisition, Resources, Validation, Writing – review & editing.

Declaration of competing interest

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

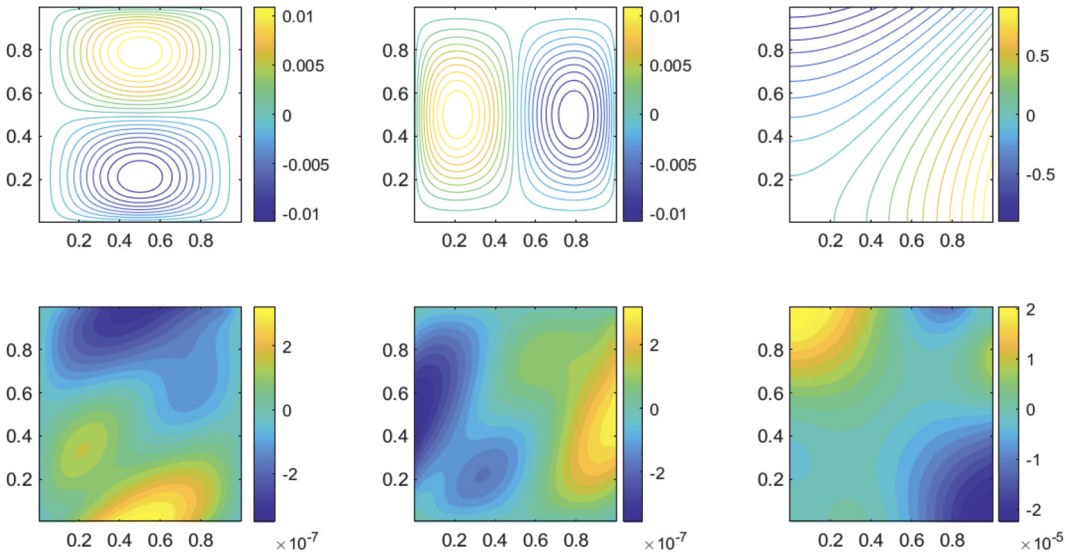

Fig. 13. The three pictures on the top row are the numerical solutions and the three at the bottom row are the error of u, v and p with $(a_x, a_y) = (100, 100)^\top$. The number of iterations is 235 and the wall time is 11.52 for Meta-MgNet.

Table 8
The mean and std of the number of iterations and the wall time of Meta-MgNet. No matter if Meta-MgNet is fine-tuned or not, the results are almost the same.

#iterations	Meta-MgNet fine-tuning	Meta-MgNet
$\epsilon = 1$	3.0 ± 0.00	4.0 ± 0.00
$\epsilon = 10^{-1}$	7.0 ± 0.00	7.5 ± 0.50
$\epsilon = 10^{-2}$	32.7 ± 0.90	35.1 ± 1.04
$\epsilon = 10^{-3}$	192.7 ± 4.29	171.6 ± 6.34
$\epsilon = 10^{-4}$	352.2 ± 7.60	375.2 ± 5.88
wall time		
$\epsilon = 1$	0.03 ± 0.00	0.03 ± 0.00
$\epsilon = 10^{-1}$	0.05 ± 0.00	0.05 ± 0.00
$\epsilon = 10^{-2}$	0.21 ± 0.00	0.22 ± 0.01
$\epsilon = 10^{-3}$	1.18 ± 0.03	1.06 ± 0.04
$\epsilon = 10^{-4}$	2.16 ± 0.05	2.31 ± 0.03

Acknowledgement

Yuyan Chen was supported in part by the PSU-PKU Joint Center for Computational Mathematics and Applications, Bin Dong by the National Natural Science Foundation of China (grant No. 11831002), Beijing Natural Science Foundation (grant No. 180001), and Beijing Academy of Artificial Intelligence (BAAI), and Jinchao Xu by the Verne M. William Professorship Fund from the Pennsylvania State University and the National Science Foundation (grant No. DMS-1819157).

Appendix A

In the appendix, we add more numerical experiments to support some of our hyper-parameter choices. We use interpolation of 2D anisotropic diffusion equations as an example, and the setting of these experiments is the same as section 5.1.

A.1. Appendix I

The **Table 8** shows the efficiency of fine-tuning for Meta-MgNet. We can find that the parameters inferred by meta smoother B_{sc} are good enough so that the result is almost the same after fine-tuning. Therefore, we can skip the fine-tuning stage.

Table 9
The mean and std of the numbers of iteration and the wall time of different choices of $\nu_1, ..., \nu_J$. The parameters of the 2D anisotropic diffusion equation are $\epsilon = 10^{-2}, \theta = 0$.

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η
$(\nu_1, ..., \nu_J) = (1, 1, 1, 1, 1)$	42.1 ± 1.58	356.6 ± 1.36	230.60 ± 6.99
$(\nu_1, ..., \nu_J) = (2, 1, 1, 1, 1)$	35.1 ± 1.04	178.9 ± 2.74	149.7 ± 3.44
$(\nu_1, ..., \nu_J) = (3, 1, 1, 1, 1)$	21.9 ± 1.81	210.90 ± 4.23	198.60 ± 6.89
$(\nu_1, ..., \nu_J) = (2, 2, 1, 1, 1)$	22.90 ± 0.70	214.50 ± 2.42	199.00 ± 4.75
wall time			
$(\nu_1, ..., \nu_J) = (1, 1, 1, 1, 1)$	0.23 ± 0.01	0.91 ± 0.01	0.59 ± 0.02
$(\nu_1, ..., \nu_J) = (2, 1, 1, 1, 1)$	0.22 ± 0.01	$\mathbf{0.44 \pm 0.01}$	$\mathbf{0.37 \pm 0.01}$
$(\nu_1, ..., \nu_J) = (3, 1, 1, 1, 1)$	$\mathbf{0.16} \pm 0.01$	0.56 ± 0.01	0.52 ± 0.02
$(\nu_1, ..., \nu_J) = (2, 2, 1, 1, 1)$	0.17 ± 0.00	0.56 ± 0.01	0.53 ± 0.01

Table 10
The mean and std of the number of iterations and the wall time of different choices of $\nu_1, ..., \nu_J$. The parameters of the 2D anisotropic diffusion equation are $\epsilon = 10^{-4}, \theta = 0$.

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η
$(\nu_1, ..., \nu_J) = (1, 1, 1, 1, 1)$	1.2e3 $\pm$ 22.51	–	6.9e3 $\pm$ 140.05
$(\nu_1, ..., \nu_J) = (2, 1, 1, 1, 1)$	375.2 ± 5.88	–	3.1e3 $\pm$ 35.70
$(\nu_1, ..., \nu_J) = (3, 1, 1, 1, 1)$	355.30 ± 10.51	–	3.6e3 $\pm$ 40.44
$(\nu_1, ..., \nu_J) = (2, 2, 1, 1, 1)$	339.00 ± 12.70	–	2.8e3 $\pm$ 33.22
wall time			
$(\nu_1, ..., \nu_J) = (1, 1, 1, 1, 1)$	6.36 ± 0.11	–	17.52 ± 0.34
$(\nu_1, ..., \nu_J) = (2, 1, 1, 1, 1)$	$\mathbf{2.31 \pm 0.03}$	–	$\mathbf{7.69 \pm 0.13}$
$(\nu_1, ..., \nu_J) = (3, 1, 1, 1, 1)$	2.50 ± 0.07	–	9.42 ± 0.11
$(\nu_1, ..., \nu_J) = (2, 2, 1, 1, 1)$	2.34 ± 0.09	–	8.45 ± 0.10

Table 11
The mean and std of the number of iterations and the wall time for the convolutional smoother B_{d} and the SC method B_{sc}.

#iterations	B_{d}	B_{sc}
$\epsilon = 1$	–	4.0 ± 0.00
$\epsilon = 10^{-1}$	58.9 ± 0.30	7.5 ± 0.50
$\epsilon = 10^{-2}$	597.8 ± 8.08	35.1 ± 1.04
$\epsilon = 10^{-3}$	5.5e3 $\pm$ 60.89	171.6 ± 6.34
$\epsilon = 10^{-4}$	–	375.2 ± 5.88
wall time		
$\epsilon = 1$	–	0.03 ± 0.00
$\epsilon = 10^{-1}$	0.21 ± 0.00	0.05 ± 0.00
$\epsilon = 10^{-2}$	2.07 ± 0.03	0.22 ± 0.01
$\epsilon = 10^{-3}$	19.04 ± 0.20	1.06 ± 0.04
$\epsilon = 10^{-4}$	–	2.31 ± 0.03

A.2. Appendix II

In section 5.1, we choose $(\nu_1, ..., \nu_J) = (2, 1, 1, 1, 1)$. Now, we compare the result of several groups of $\nu_1, ..., \nu_J$. Since it is easier to estimate error on coarse grid, to set $\nu_i > 1, i = 3, 4, 5$ is not necessary. Thus, we only test some pair of ν_1 and ν_2, and set $\nu_i = 1$ for $i = 3, 4, 5$. The numerical experiments in **Tables 9 and 10** show $(\nu_1, ..., \nu_J) = (2, 1, 1, 1, 1)$ is a better choice.

A.3. Appendix III

We compare the efficiency between B_{d} in **Algorithm 3** and B_{sc} in **Table 11**. As we can see, B_{sc} is much better than B_{d}. Therefore, we choose B_{sc} as the meta smoother in other numerical experiments in this paper.

References

[1] C. Audouze, F. De Vuyst, P. Nair, Reduced-order modeling of parameterized pdes using time-space-parameter principal component analysis, Int. J. Numer. Methods Eng. 80 (2009).
[2] R.E. Bank, C.C. Douglas, Sharp estimates for multigrid rates of convergence with general smoothing and acceleration, SIAM J. Numer. Anal. 22 (1985) 617–633.
[3] C. Beck, W. E, A. Jentzen, Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations, J. Nonlinear Sci. 29 (2019) 1563–1619, https://doi.org/10.1007/s00332-018-9525-3.
[4] P. Benner, S. Gugercin, K. Willcox, A survey of projection-based model reduction methods for parametric dynamical systems, SIAM Rev. 57 (2015) 483–531, https://doi.org/10.1137/130932715.
[5] A. Brock, T. Lim, J.M. Ritchie, N. Weston, SMASH: one-shot model architecture search through hypernetworks, CoRR abs/1708.05344, http://arxiv.org/abs/1708.05344, arXiv:1708.05344, 2017.
[6] L. Chen, M. Wang, L. Zhong, Convergence analysis of triangular mac schemes for two dimensional Stokes equations, J. Sci. Comput. 63 (2015) 716–744.
[7] T.Q. Chen, Y. Rubanova, J. Bettencourt, D.K. Duvenaud, Neural ordinary differential equations, in: Advances in Neural Information Processing Systems, 2018, pp. 6571–6583.
[8] Y. Chen, W. Yu, T. Pock, On learning optimized reaction diffusion processes for effective image restoration, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
[9] N. Discacciati, J.S. Hesthaven, D. Ray, Mathicse technical report: controlling oscillations in high-order discontinuous Galerkin schemes using artificial viscosity tuned by neural networks, http://infoscience.epfl.ch/record/263616, 2019, 10.5075/epfl-MATHICSE-263616.
[10] W. E, A proposal on machine learning via dynamical systems, Commun. Math. Stat. 5 (2017) 1–11.
[11] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12, https://doi.org/10.1007/s40304-018-0127-z.
[12] J. Feliu-Faba, Y. Fan, L. Ying, Meta-learning pseudo-differential operators with deep neural networks, arXiv e-prints, arXiv:1906.06782, 2019.
[13] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, JMLR.org, 2017, pp. 1126–1135.
[14] S. Fresca, L. Dede, A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes, arXiv:2001.04001, 2020.
[15] D. Greenfeld, M. Galun, R. Kimmel, I. Yavneh, R. Basri, Learning to optimize multigrid pde solvers, arXiv:1902.10248, 2019.
[16] K. Gregor, Y. LeCun, Learning fast approximations of sparse coding, in: Proceedings of the 27th International Conference on International Conference on Machine Learning, Omnipress, 2010, pp. 399–406.
[17] D. Ha, A.M. Dai, Q.V. Le, Hypernetworks, CoRR abs/1609.09106, http://arxiv.org/abs/1609.09106, arXiv:1609.09106, 2016.
[18] E. Haber, L. Ruthotto, Stable architectures for deep neural networks, Inverse Probl. 34 (2017) 014004.
[19] W. Hackbusch, Multi-Grid Methods and Applications, vol. 4, Springer Science & Business Media, 2013.
[20] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci 115 (2018) 8505–8510, https://doi.org/10.1073/pnas.1718942115, https://www.pnas.org/content/115/34/8505, arXiv: https://www.pnas.org/content/115/34/8505.full.pdf.
[21] J. He, J. Xu, Mgnet: a unified framework of multigrid and convolutional neural network, CoRR abs/1901.10415, http://arxiv.org/abs/1901.10415, arXiv:1901.10415, 2019.
[22] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[23] G. Hinton, L. Deng, D. Yu, G. Dahl, A.r.Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, T. Sainath, Deep neural networks for acoustic modeling in speech recognition, IEEE Signal Process. Mag. 29 (2012) 82–97, https://www.microsoft.com/en-us/research/publication/deep-neural-networks-for-acoustic-modeling-in-speech-recognition/.
[24] T. Hospedales, A. Antoniou, P. Micaelli, A. Storkey, Meta-learning in neural networks: a survey, arXiv:2004.05439, 2020.
[25] J.T. Hsieh, S. Zhao, S. Eismann, L. Mirabella, S. Ermon, Learning neural PDE solvers with convergence guarantees, arXiv e-prints, arXiv:1906.01200, 2019.
[26] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: CVPR, 2017, p. 3.
[27] A. Katrutsa, T. Daulbaev, I. Oseledets, Deep multigrid: learning prolongation and restriction matrices, arXiv preprint, arXiv:1711.03825, 2017.
[28] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436.
[29] K. Lee, K.T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020) 108973, https://doi.org/10.1016/j.jcp.2019.108973, http://www.sciencedirect.com/science/article/pii/S0021999119306783.
[30] Y. Li, J. Lu, A. Mao, Variational training of neural network approximations of solution maps for physical models, arXiv e-prints, arXiv:1905.02789, 2019.
[31] Z. Liu, W. Cai, Z.Q.J. Xu, Multi-scale deep neural network (mscalednn) for solving Poisson-Boltzmann equation in complex domains, arXiv preprint, arXiv:2007.11207, 2020.
[32] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.T. Cheng, J. Sun, Metapruning: meta learning for automatic neural network channel pruning, in: The IEEE International Conference on Computer Vision (ICCV), 2019.
[33] J. Lorraine, D. Duvenaud, Stochastic hyperparameter optimization through hypernetworks, CoRR abs/1802.09419, http://arxiv.org/abs/1802.09419, arXiv:1802.09419, 2018.
[34] Y. Lu, A. Zhong, Q. Li, B. Dong, Beyond finite layer neural networks: bridging deep architectures and numerical differential equations, in: Proceedings of the 35th International Conference on Machine Learning, PMLR, 2018.
[35] I. Luz, M. Galun, H. Maron, R. Basri, I. Yavneh, Learning algebraic multigrid using graph neural networks, arXiv:2003.05744, 2020.
[36] J. Magiera, D. Ray, J.S. Hesthaven, C. Rohde, Constraint-aware neural networks for Riemann problems, arXiv e-prints, arXiv:1904.12794, 2019.
[37] A. Nichol, J. Achiam, J. Schulman, On first-order meta-learning algorithms, CoRR abs/1803.02999, http://arxiv.org/abs/1803.02999, arXiv:1803.02999, 2018.
[38] R. Nicolaides, X. Wu, Analysis and convergence of the mac scheme. ii. Navier-Stokes equations, Math. Comput. 65 (1996) 29–44.
[39] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations, arXiv e-prints, arXiv:1711.10561, 2017.
[40] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, arXiv e-prints, arXiv:1711.10566, 2017.
[41] D. Ray, J.S. Hesthaven, Detecting troubled-cells on two-dimensional unstructured grids using a neural network, J. Comput. Phys. 397 (2019) 108845, https://doi.org/10.1016/j.jcp.2019.07.043, http://www.sciencedirect.com/science/article/pii/S0021999119305297.
[42] H. Sheng, C. Yang, Pfnn: a penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries, arXiv preprint, arXiv:2004.06490, 2020.
[43] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (2016) 484.
[44] J. Sirignano, K. Spiliopoulos, Dgm: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364, https://doi.org/10.1016/j.jcp.2018.08.029, http://www.sciencedirect.com/science/article/pii/S0021999118305527.

[45] S. Thrun, L. Pratt, Learning to learn: introduction and overview, in: Learning to Learn, 1998.
[46] Z. Wang, Z. Zhang, A mesh-free method for interface problems using the deep learning approach, J. Comput. Phys. 400 (2020) 108963, https://doi.org/10.1016/j.jcp.2019.108963, http://www.sciencedirect.com/science/article/pii/S0021999119306680.
[47] H. Xu, H. Chang, D. Zhang, Dlga-pde: discovery of pdes with incomplete candidate library via combination of deep learning and genetic algorithm, J. Comput. Phys. 109584 (2020).
[48] J. Xu, Theory of Multilevel Methods, volume 8924558 Cornell University, Ithaca, NY, 1989.
[49] J. Xu, Iterative methods by space decomposition and subspace correction, SIAM Rev. 34 (1992) 581–613.
[50] J. Xu, L. Zikatanov, The method of alternating projections and the method of subspace corrections in Hilbert space, J. Am. Math. Soc. 15 (2002) 573–597.
[51] J. Xu, L. Zikatanov, Algebraic multigrid methods, Acta Numer. 26 (2017) 591–721.
[52] G. Yu, J. Xu, L.T. Zikatanov, Analysis of a two-level method for anisotropic diffusion equations on aligned and nonaligned grids, Numer. Linear Algebra Appl. 20 (2013) 832–851.
[53] D. Zhang, L. Guo, G.E. Karniadakis, Learning in modal space: solving time-dependent stochastic pdes using physics-informed neural networks, SIAM J. Sci. Comput. 42 (2020) A639–A665, https://doi.org/10.1137/19M1260141.
[54] H. Zhang, B. Liu, H. Yu, B. Dong, Metainv-net: meta inversion network for sparse view ct image reconstruction, arXiv:2006.00171, 2020.