arXiv:2111.08823v1 [cs.LG] 15 Nov 2021

Meta-Auto-Decoder for Solving Parametric Partial Differential Equations

Xiang Huang[1], Zhanhong Ye[4], Hongsheng Liu[2], Beiji Shi[2], Zidong Wang[2], Kang Yang[2], Yang Li[2], Bingya Weng[2], Min Wang[2], Haotian Chu[2], Jing Zhou[2], Fan Yu[2], Bei Hua[1], Lei Chen[3], Bin Dong[4*]

[1]University of Science and Technology of China, [2]Huawei Technologies Co. Ltd

[3]Hong Kong University of Science and Technology, [4]Peking University

[1]sahx@mail.ustc.edu.cn, bhua@ustc.edu.cn

[2]liuhongsheng4, shibeiji, wang1, yangkang22, liyang477@huawei.com

[2]wengbingya, wangmin106, chuhaotian2, zhoujing3, fan.yu@huawei.com

[3]leichen@cse.ust.hk,

[4]yezhanhong@pku.edu.cn, dongbin@math.pku.edu.cn

Abstract

Partial Differential Equations (PDEs) are ubiquitous in many disciplines of science and engineering and notoriously difficult to solve. In general, closed-form solutions of PDEs are unavailable and numerical approximation methods are computationally expensive. The parameters of PDEs are variable in many applications, such as inverse problems, control and optimization, risk assessment, and uncertainty quantification. In these applications, our goal is to solve parametric PDEs rather than one instance of them. Our proposed approach, called Meta-Auto-Decoder (MAD), treats solving parametric PDEs as a meta-learning problem and utilizes the Auto-Decoder structure in [1] to deal with different tasks/PDEs. Physics-informed losses induced from the PDE governing equations and boundary conditions is used as the training losses for different tasks. The goal of MAD is to learn a good model initialization that can generalize across different tasks, and eventually enables the unseen task to be learned faster. The inspiration of MAD comes from (conjectured) low-dimensional structure of parametric PDE solutions and we explain our approach from the perspective of manifold learning. Finally, we demonstrate the power of MAD though extensive numerical studies, including Burgers' equation, Laplace's equation and time-domain Maxwell's equations. MAD exhibits faster convergence speed without losing the accuracy compared with other deep learning methods.

1 Introduction

Partial differential equations (PDEs) play a significant role in scientific research and engineering design. Many physical problems such as propagation of sound, fluid flow, and electrodynamics fields, are mathematically formulated by PDEs. A lot of applications require to solve the PDEs with different physical parameters, boundary conditions and even solution regions, such as inverse problems, control and optimization, risk assessment, and uncertainty quantification [2, 3]. Mathematically, it requires to solve the so-called *parametric* PDEs that can be formally formulated as:

$$\begin{aligned} \mathcal{L}_x^\eta u &= \phi_\eta, \quad x \in \Omega \subset \mathbb{R}^d \\ \mathcal{B}_x^\eta u &= 0, \qquad x \in \partial\Omega \end{aligned} \tag{1}$$

where $\mathcal{L}^\eta$ is a partial differential operator parametrized by η and x denote the independent variable in spatiotemporal-dependent PDEs. Given $\mathcal{A} = \mathcal{A}(\Omega; \mathbb{R}^{d_\eta})$ and $\mathcal{U} = \mathcal{U}(\Omega; \mathbb{R}^{d_u})$, $\eta \in \mathcal{A}$ is the variable parameter of the PDEs and $u \in \mathcal{U}$ is the solution of the PDEs.

*Corresponding author.

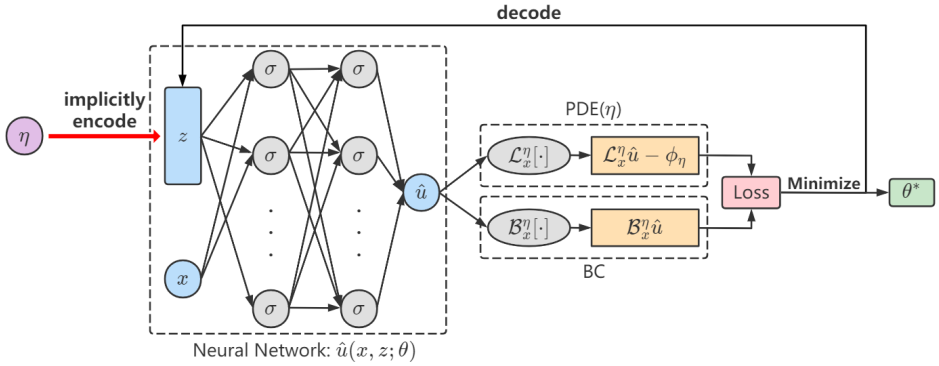


Figure 1: Schematic of Meta-Auto-Decoder.

Traditional solvers, e.g., finite element methods (FEM) [4] or finite difference methods (FDM) [5], require prohibitively long simulation time as they need to run independent simulation for each η. Solving parametric PDEs requires to learn infinite-dimensional operators that map any PDE parameter η to its corresponding solution u^η (i.e., the parameter-to-solution mapping). Recently, a series of neural operator methods are proposed to learn infinite-dimensional operators [6, 7, 8, 9]. These methods remedy the mesh-dependent nature of classical numerical solvers and moreover they require to train the neural network only once. Although these methods have demonstrated promising results across a wide range of applications, they require a large corpus of paired input-output observations that involve a large number of expensive numerical simulations. In addition, these methods can only return a coarse approximation to the unknown solution operator and there is no guarantee that the predicted outputs satisfy the PDEs.

Recently a few neural approximation methods [10, 11, 12, 13] are developed to solve the fixed-parameter PDEs using deep neural networks, where the loss functions are based on physical laws. These methods can work without any labeled data, although labeled data can be used as a loss penalty item. However, similar to traditional numerical methods, all these methods have to retrain the neural networks for different PDE parameters. In other words, these methods treat different PDE parameters as different tasks, and each task requires to train the deep neural networks from scratch, which is quite expensive and impractical.

Our goal in this paper is to provide a new unsupervised approach to tackle parametric PDEs using deep neural networks. Unlike previously mentioned methods, we treat solving parametric PDEs as a meta-learning problem. More specifically, an individual PDE with the parameter η is viewed as a single task with η drawn from the distribution of tasks $p_{\mathcal{A}}$. Here, we loosely define a task to be the PDE parameter and the physics-informed loss function [10, 11] $\mathcal{T} = \{\eta, L\}$. Our method aims to learn a deep neural network backbone θ that can generalize across different tasks, and eventually enable the unseen task to be learned faster. The proposed Meta-Auto-Decoder approach in Fig.1 utilizes the concept of auto-decoder in [1] and consists of two stages: pre-training stage and fine-tuning stage. During the pre-training stage, our approach implicitly encodes each task $\mathcal{T}$ by a trainable latent vector z. Different tasks share the same neural network backbone while the input is the concatenation of the latent vector z and data point x from each task. Both the latent vector and neural network model are updated during the pre-training stage. In the end, a shared deep neural network backbone is learned for all tasks and each task is paired with its own decoded latent vector. Such backbone is often considered as the *meta knowledge* as it benefits from all tasks and the learned latent vector z is the *task-specific knowledge*. We interpret the pre-trained backbone as the learned low-dimensional manifold in the high-dimensional space and the latent vector z defines the coordinates on the manifold. When dealing a new task, the proposed method encodes it through a new latent vector z and fine-tunes z and θ with initialization from the pre-trained model to refine both the manifold and the coordinates on the manifold. The proposed Meta-Auto-Decoder is closed related to the DeepSDF proposed in [1]. However, in our work, we introduce a manifold learning understanding of solving parametric PDEs in general which not only automatically grants a manifold learning view of [1], but also leads to the new method MAD. Unlike the DeepSDF which only refines the latent vector z, MAD refines both θ and z as originating naturally from the manifold understanding. Moreover, MAD uses mesh-free representation and an unsupervised loss which is more flexible in practice.

To summarize, we list our contributions as follows:

- We propose a novel mesh-independent and label-free deep learning method, Meta-Auto-Decoder, to solve parametric PDEs.
- We explain the effectiveness of MAD from a manifold learning perspective. In the pre-training stage, MAD approximates the solution manifold with a deep neural network. In the fine-tuning stage, we search the approximate solution corresponding to the new PDE parameter in the pre-trained manifold by fine-tuning the latent vector only.
- In some cases, the exact solutions cannot be expressed as a specific manifold in the function space, but lie in the neighborhood of a specific manifold. Therefore, it is not enough to fine-tune the latent vector only, and we have to fine-tune the model weight with initialization θ^*. Because the latent vector lies in a low-dimensional space and the model weight are initialized with pre-trained weight θ^*, MAD is expected to converge faster in the fine-tuning stage.
- The effectiveness of MAD is demonstrated through numerical experiments. These results show that the MAD method significantly improves the convergence speed in the fine-tuning stage compared with the other deep learning methods.

The rest of this paper is organized as follows: we introduce some related works in Sec.2. Then, we explain the MAD method in detail to deal with parametric PDEs in Sec.3. The extensive experiments demonstrate the effectiveness of the MAD method with various parametric PDEs in Sec.4. Finally, we summarize our work and look forward to future work in Sec.5.

2 Related works

In this section, we review the deep learning tools for solving (parametric) PDEs. There are mainly three lines of work on this direction: finite-dimensional operators, neural approximations and neural operators. In addition, we also briefly discuss the related meta-learning algorithms.

2.1 Finite-dimensional operators

This kind of approaches use a deep convolutional neural network to learn the solution mapping of parametric PDEs between two finite-dimensional Euclidean spaces [14, 15, 16, 17]. For example, Galerkin Transformer [17] is an attention operator that can achieve highly accurate PDE solution mappings. Due to the model structure, these approaches are mesh-dependent and require model tuning for different resolution. Furthermore, these methods are limited to domain/geometry discretization and therefore, one cannot query solutions at any other points different from the mesh. In contrast, our method is mesh independent and can predict any points inside the domain.

2.2 Neural Operators

Different from classical numerical methods that solve one instance of the PDEs at a time, this kind of approaches use neural networks to directly learn mesh-free, infinite-dimensional operator for parametric PDEs that is a mapping between two infinite-dimensional function spaces [6, 7, 8, 18, 19]. Bhattacharya et al. [7] uses principal component analysis in Hilbert space to finite-dimensionalize the input and output function spaces, and then use a neural network to establish the mapping between the two finite-dimensional spaces. Lu et al. [6] propose DeepONet network architecture that uses two subnets to encode the parameters and location variables of the PDEs separately, and then merge them together to compute the solution. Fourier Neural Operator (FNO) [8] utilizes fast Fourier transform (FFT) to build neural operator architecture and learn the mapping between two infinite-dimensional function spaces. The advantage of these approaches is that once the neural network is trained, the prediction time is almost negligible. However, the data acquisition cost is prohibitive in complex physical, biological, or engineering systems, and the generalization ability of the model (from one PDE parameter to another) is poor when few labeled data are available [20].

2.3 Neural Approximations

This kind of approaches mainly rely on governing equations and boundary conditions (or their variants) to train the neural networks. For example, Physics-Informed Neural Networks (PINNs) [10] and Deep Galerkin Method (DGM) [11] constrain the output of deep neural networks to satisfy the given governing equations and boundary conditions. Weinan et al. [12] propose Deep Ritz Method (DRM) that exploits the variational form of PDEs and can only be used to solve PDEs that can be reformulated as equivalent energy minimization problems. Zang et al. [13] propose Weak Adversarial Network (WAN) that based on a weak formulation of PDEs parameterizes the weak solution and test functions as primal and adversarial neural networks, respectively. These neural approximation methods can work in an unsupervised manner, without the need to generate labeled data from conventional computational methods. However, none of them can be used to solve parametric PDEs as they have to retrain the neural network under different PDE parameters. Weinan et al. [12] recommend transfer learning to mitigate retraining cost by using the model weight trained for one task as the model initialization for training another task. However, no obvious gain is observed when the correlation between the two tasks is small, which is the biggest limitation of the transfer learning method.

2.4 Meta-Learning

Recently, the field of meta-learning has attracted dramatically rising attention in the machine learning community. Different from the conventional AI approaches which aim to solve tasks from scratch using a fixed algorithm, meta-learning targets on improving the learning algorithms themselves based on multiple learning episodes. A comprehensive survey on this direction can be found in [21]. Closely related to this work, the model-agnostic gradient-based meta learning algorithms [22, 23, 24, 25] seek an initialization of model weight such that it can be adapted to a new task with a few number of gradient updates and produce better generalization. For example, the model-agnostic meta learning (MAML) [22] trains for few-shot generalization by optimizing for a set of initial model weight θ such that one or a few steps of gradient descent on the training dataset achieves good performance on the test dataset. The Reptile [24] algorithm works by training on the sampled task and updating the initialization θ towards the new weight for the selected task. The MetaSDF [26] utilizes the gradient-based meta learning to effectively learn a prior over implicit neural representations for signed distance functions. Since parametric PDEs can be formulated from a meta-learning perspective, we can adapt these meta-learning methods to solve the PDE problems. However, for PDEs that are difficult to train, these meta-learning methods do not bring significant gains in convergence speed. In Sec.4, we test the performance of these methods for solving parametric PDEs as a comparison.

3 Methodology

In this section, we will first review the physics-informed losses induced by parametric PDEs and then introduce our MAD method to learn the solution operator mapping for parametric PDEs in detail.

3.1 Physics-Informed Loss

Given any PDE parameter $\eta \in \mathcal{A}$, we denote the physics-informed loss $L^\eta : \mathcal{U} \to [0, \infty)$ for Eq.(1) by

$$L^\eta[u] = L^\eta_{\mathrm{r}}[u] + \lambda_{\mathrm{bc}} L^\eta_{\mathrm{bc}}[u], \tag{2}$$

where

$$L^\eta_{\mathrm{r}}[u] = \|\mathcal{L}^\eta_x u - \phi_\eta\|^2_{L^2(\Omega)} \tag{3}$$

is the loss corresponding to the PDE residual,

$$L^\eta_{\mathrm{bc}}[u] = \|\mathcal{B}^\eta_x u\|^2_{L^2(\partial\Omega)} \tag{4}$$

is the loss related to the boundary condition (the initial condition is a special boundary condition), and $\lambda_{\mathrm{bc}} > 0$ is weighting coefficient.

It is obvious that $u \in \mathcal{U}$ solves the Eq.(1) if and only if $L^\eta[u] = 0$ holds. We assume that for any $\eta \in \mathcal{A}$, such an exact solution always exists and is unique, and we denote it as u^η. Then a solution mapping [1] is defined as follows:

$$\begin{aligned} G : \mathcal{A} &\to \mathcal{U}, \\ \eta &\mapsto G(\eta) = u^\eta. \end{aligned} \tag{5}$$

To compute $G(\eta)$, i.e. getting the solution u^η for given parameter $\eta \in \mathcal{A}$, the PINNs method [27] tries to approximate $u^\eta(x)$ by a neural network $u(x;\theta)$, and find $\theta \in \Theta$ by solving the optimization problem

$$\arg\min_\theta \hat{L}^\eta[u(x;\theta)], \tag{6}$$

where

$$\begin{aligned} \hat{L}^\eta[u] &= \hat{L}^\eta_{\mathrm{r}}[u] + \lambda_{\mathrm{bc}} \hat{L}^\eta_{\mathrm{bc}}[u] \\ &= \frac{1}{M_{\mathrm{r}}} \sum_{j=1}^{M_{\mathrm{r}}} \left\| \mathcal{L}^\eta_x u(x^{\mathrm{r}}_j) - \phi_\eta(x^{\mathrm{r}}_j) \right\|_2^2 + \frac{\lambda_{\mathrm{bc}}}{M_{\mathrm{bc}}} \sum_{j=1}^{M_{\mathrm{bc}}} \left\| \mathcal{B}^\eta_x u(x^{\mathrm{bc}}_j) \right\|_2^2 \end{aligned} \tag{7}$$

is the Monte Carlo estimate of $L^\eta[u]$, and $\mathcal{D}_{\mathrm{r}} = \{x^{\mathrm{r}}_j\}_{j\in\{1,\dots,M_{\mathrm{r}}\}}$ and $\mathcal{D}_{\mathrm{bc}} = \{x^{\mathrm{bc}}_j\}_{j\in\{1,\dots,M_{\mathrm{bc}}\}}$ are random sampling points of Ω and $\partial\Omega$, respectively.

3.2 The Low-Dimensional Assumption

The parametric PDEs Eq.(1) allows the parameter η to vary within a certain range. Training a specific neural network for each η from scratch is neither fast nor very useful in practice. Instead, we aim to build a model that can quickly obtain the PDE solution for any given η with fewer training iterations. Considering the relations among these parametric PDEs, we make the following assumption on the solution mapping G:

Assumption 1 *The set of PDE solutions $G(\mathcal{A}) = \{G(\eta) \mid \eta \in \mathcal{A}\} \subset \mathcal{U}$ is contained in some set with relatively low-dimensional structure. To be more specific, we assume that there is a finite-dimensional space $Z = \mathbb{R}^l$ (with $l \ll \dim\mathcal{U}$) and a Lipschitz continuous mapping*

$$\bar{G} : Z \to \mathcal{U}$$

such that $G(\mathcal{A}) \subseteq \bar{G}(Z)$. In other words, for any $\eta \in \mathcal{A}$, there exists some $z \in Z$ satisfying $\bar{G}(z) = G(\eta)$.

We have the following comments on the above assumption:

- The mapping $\bar{G}$ is Lipschitz continuous if and only if there exists some $C > 0$ such that $\left\|\bar{G}(z) - \bar{G}(z')\right\|_{\mathcal{U}} \le C\|z - z'\|$ for all $z, z' \in Z$. This excludes mappings that are highly irregular.
- The assumption is obviously satisfied when $\mathcal{A}$ is itself a finite-dimensional space and G is Lipschitz continuous (just take $Z = \mathcal{A}$, $\bar{G} = G$).
- Under certain smoothness and non-degeneracy conditions on $\bar{G}$, the subset $\bar{G}(Z) \subset \mathcal{U}$ forms an embedded submanifold, and our method presented below can be viewed as a manifold-learning approach.
- Since $\dim Z \ll \dim\mathcal{U}$ (the latter is usually infinity) holds, we may view the mapping $\bar{G}$ as some sort of "decoder", and Z the corresponding latent vector space, despite of the fact that there doesn't exist an "encoder".

[1] In the literature, we would call G a "solution operator" if $\mathcal{A} = \mathcal{A}(\Omega;\mathbb{R}^{d_\eta})$ is a separable Banach space of functions taking values in $\mathbb{R}^{d_\eta}$. A more general choice of $\mathcal{A}$ is allowed in this paper, for example a subset of some linear space, or a discrete set.

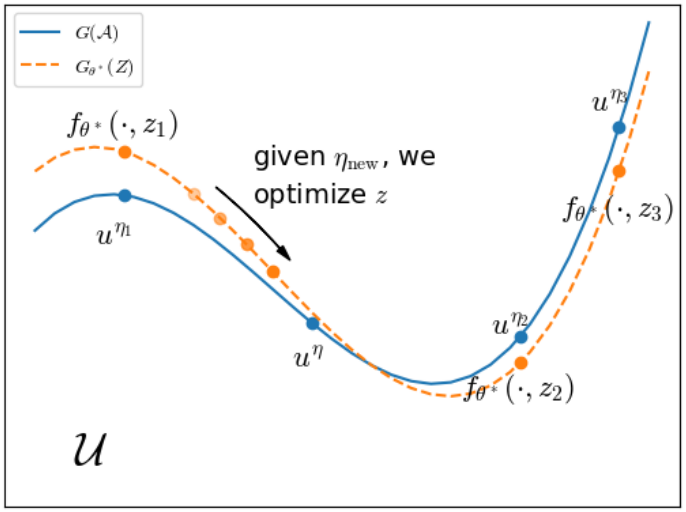


Figure 2: Illustration of how MAD works. Given PDE parameter $\eta_{\text{new}} \in \mathcal{A}$, we want to find its solution $u^{\eta_{\text{new}}} = G(\eta_{\text{new}})$ which is a point in the function space $\mathcal{U}$. Solutions for all possible PDE parameters form a set $G(\mathcal{A})$ with low-dimensional structure, whose approximate parametrization is given by the MAD $G_{\theta^*}(z) = f_{\theta^*}(\cdot, z)$, and each η_i implicitly corresponds to some $z_i \in Z$. We then search for the solution $u^{\eta_{\text{new}}}$ in this parametrized space, i.e. optimize $z \in Z$.

If we have a decoder mapping $\bar{G}$ as above, then for a given parameter η, we no longer need to search for the solution u^η in the whole space $\mathcal{U}$. Instead, we may focus on the smaller subset $\bar{G}(Z)$, i.e. the class of functions in $\mathcal{U}$ that is parametrized by Z, since $u^\eta = G(\eta) \in \bar{G}(Z)$ holds for any $\eta \in \mathcal{A}$. We then solve the optimization problem

$$z^\eta = \arg\min_z L^\eta[\bar{G}(z)], \tag{8}$$

and $\bar{G}(z^\eta)$ is the approximate solution.

3.3 Meta-Auto-Decoder

Now assuming that the dimension of Z is chosen (either empirically or according to theoretical analysis), we try to find the mapping $\bar{G}$. Since such a decoder mapping is usually complex and hard to design by hand, we consider the θ-parametrized version[2]

$$G_\theta : Z \to \mathcal{U},$$

and find the best θ automatically by solving an optimization problem. In fact, this is what auto-decoder proposed in [1] actually does. We use a neural network to represent G_θ:

$$G_\theta(z)(x) = f_\theta(x, z), \tag{9}$$

where f_θ is a neural network whose input is the concatenation of $x \in \mathbb{R}^d$ and $z \in \mathbb{R}^l$.

Subsequently, we can find the optimal network weight θ via training, with the target being $G(\mathcal{A}) \subseteq G_\theta(Z)$. Assuming that the PDE parameters are generated from a probability distribution $\eta \sim p_{\mathcal{A}}$, then $G(\eta) \in G_\theta(Z)$ holds almost surely if and only if

$$d(\theta) = \mathop{\mathbb{E}}_{\eta \sim p_{\mathcal{A}}} \left[d_{\mathcal{U}}\big(u^\eta, G_\theta(Z)\big) \right] = \mathop{\mathbb{E}}_{\eta \sim p_{\mathcal{A}}} \left[\min_z \left\| u^\eta - f_\theta(\cdot, z) \right\|_{\mathcal{U}} \right] = 0, \tag{10}$$

which suggests taking $\theta^* = \arg\min_\theta d(\theta)$. In case we do not have direct access to the exact solutions u^η, the equivalent condition

$$d'(\theta) = \mathop{\mathbb{E}}_{\eta \sim p_{\mathcal{A}}} \left[\min_z L^\eta[f_\theta(\cdot, z)] \right] = 0 \tag{11}$$

[2]For fixed θ, the set $G_\theta(Z)$ is parametrized by $z \in Z$. When θ changes, this set changes accordingly. In some sense, θ parametrizes a parametrization.

is considered, and we search for the minimizer of $d'(\theta)$ instead.

In the specific implementation, the expectation on $\eta \sim p_{\mathcal{A}}$ is estimated by Monte Carlo samples $\eta_1, \ldots, \eta_N$, and the optimal network weight θ is taken to be

$$\theta^* \approx \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \min_{z_i} L^{\eta_i}[f_\theta(\cdot, z_i)]. \tag{12}$$

We further estimate the physics-informed loss L^η using random sampling points. To sum up, our MAD method has the following two stages:

Pre-training Stage Given N randomly generated PDE parameters $\eta_1, \ldots, \eta_N \in \mathcal{A}$ and sampling points $\mathcal{D}_{\mathrm{r}}^i = \{x_{i,j}^{\mathrm{r}}\}_{j\in\{1,\ldots,M_{\mathrm{r}}\}} \subset \Omega$, $\mathcal{D}_{\mathrm{bc}}^i = \{x_{i,j}^{\mathrm{bc}}\}_{j\in\{1,\ldots,M_{\mathrm{bc}}\}} \subset \partial\Omega$, we solve the following optimization problem

$$\arg\min_{\theta, \{z_i\}_{i\in\{1,\ldots,N\}}} \sum_{i=1}^{N} \left(\hat{L}^{\eta_i}[f_\theta(\cdot, z_i)] + \frac{1}{\sigma^2}\|z_i\|^2 \right) \tag{13}$$

to obtain the optimal θ^* and corresponding latent vectors $\{z_i^*\}_{i\in\{1,\ldots,N\}}$, where $\hat{L}^{\eta_i}$ is defined in (7). The regularization term $\frac{1}{\sigma^2}\|z_i\|^2$ is added for training stability.

Fine-tuning Stage Given a new PDE parameter η_{new} and the spatial sampling points $\mathcal{D}_{\mathrm{r}} \subset \Omega$,$\mathcal{D}_{\mathrm{bc}} \subset \partial\Omega$, we compute $G(\eta_{\mathrm{new}})$ (i.e. find the solution $u^{\eta_{\mathrm{new}}}$) as follows: keeping θ^* fixed, and optimizing the latent vector z to get

$$z_{\mathrm{new}} = \arg\min_{z} \hat{L}^{\eta}[f_{\theta^*}(\cdot, z)] + \frac{1}{\sigma^2}\|z\|^2, \tag{14}$$

then $f_{\theta^*}(x, z_{\mathrm{new}}) \approx u^{\eta_{\mathrm{new}}}(x)$ is the approximate solution for PDE parameter η_{new}. For the initialization of z in the fine-tuning stage, we can use a z_i^* obtained in the pre-training stage as the initialization of z. For the selection of z_i^*, we can use the distance[3] between η_{new} and η_i as a criterion.

An intuitive illustration of how MAD works is given in Fig.2. In fact, when we switch back to the original $\mathcal{U}$-distance against the exact solutions as in the form of Eq.(10), and add a problem-specific clamping operation, Eq.(13) and Eq.(14) would then coincide with the auto-decoder algorithm proposed in [1]. However, our derivation presented above applies to a wider range of learning problems, and is no longer limited to solving parametric PDEs.

3.4 Meta-Auto-Decoder with Fine-Tuned Model

Assumption 1 may fail in some occasions, especially when the parameter set $\mathcal{A}$ of PDEs is an infinite-dimensional function space. In this case, we consider the following weaker assumption:

Assumption 2 *The set of possible solutions $G(\mathcal{A}) \subset \mathcal{U}$ can be approximated by a set with low-dimensional structure, in the sense that there is a finite-dimensional space $Z = \mathbb{R}^l$ (with $l \ll \dim \mathcal{U}$) and a Lipschitz continuous mapping*

$$\bar{G} : Z \to \mathcal{U}$$

such that $G(\mathcal{A})$ is contained in the c-neighborhood of $\bar{G}(Z) \subset \mathcal{U}$, where c is a relatively small constant. In other words, for any $\eta \in \mathcal{A}$, there exists some $z \in Z$ satisfying $\|\bar{G}(z) - G(\eta)\|_{\mathcal{U}} \le c$.

Under this new assumption, similar derivation leads to the same pre-training stage, and we use it to find the initial decoder mapping $G_{\theta^*} \approx \bar{G}$. However, in the fine-tuning stage, simply optimizing the latent vector z won't give a satisfactory solution in general due to the existence of the c-gap. Therefore, we try to fine-tune the model weight θ with the latent vector z simultaneously, and solve the following optimization problem

$$(z_{\mathrm{new}}, \theta_{\mathrm{new}}) = \arg\min_{z,\theta} \hat{L}^{\eta}[f_\theta(\cdot, z)] + \frac{1}{\sigma^2}\|z\|^2 \tag{15}$$

with initialization $\theta = \theta^*$. This would produce a new decoder $G_{\theta_{\mathrm{new}}}$ specific to the PDE parameter η_{new}. An intuitive illustration is given in Fig.3.

[3] We can discretize a function into a vector and then find the Euclidean distance between the two vectors as the distance between two PDE parameters.

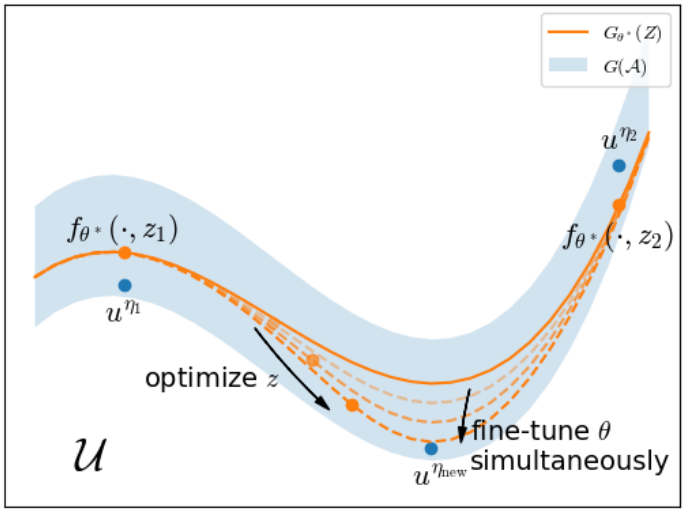


Figure 3: Illustration of how MAD with fine-tuned model works. The pre-trained MAD still parametrizes a set $G_{\theta^*}(Z)$, but this time $G(\mathcal{A})$ lies within a neighborhood of it represented by a gray shadow band in the figure. To find the solution $u^{\eta_{\text{new}}}$, we have to fine-tune θ (with initialization θ^*) simultaneously when z is being optimized.

We give an example of $G(\mathcal{A})$ which satisfies Assumption 2 but violates Assumption 1. Suppose $\mathcal{U} = L^2(\Omega)$ has an orthonormal basis $\{\phi_k(x)\}_{k\in\mathbb{N}}$, and $(\lambda_k)_{k\in\mathbb{N}}$ is a sequence of positive real numbers with $\sum_{k=1}^{\infty} \lambda_k < \infty$. Then we consider the case

$$G(\mathcal{A}) = \left\{ \sum_{k=1}^{\infty} \xi_k \sqrt{\lambda_k} \phi_k \;\middle|\; \xi_k \in [-1, 1] \right\} \subset L^2(\Omega).$$

It is easy to see that no $(Z, \bar{G})$-pair can make Assumption 1 true. As for Assumption 2, once the number c given, there exists a large enough l to make $\sum_{k=l+1}^{\infty} \lambda_k < c^2$ holds. Then we let $Z = \mathbb{R}^l$ and choose a linear mapping $\bar{G}$ such that

$$\bar{G}(Z) = \left\{ \sum_{k=1}^{l} \xi_k \sqrt{\lambda_k} \phi_k \;\middle|\; \xi_k \in \mathbb{R} \right\},$$

which completes our construction.

We may expect that such fine-tuning procedure finds the solution faster than training from scratch. The latent vector z lies in a low-dimensional space, and its optimization could be as simple as before. The model weight θ are high-dimensional, but already with a good initialization $\theta = \theta^*$, and its fine-tuning only needs to deal with the small c-gap, hence the optimization is expected to be simple as well. Performance of MAD is examined experimentally in Sec.4.

3.5 A Motivational Example and Visualization

Intuitively, as θ changes, the set $G_\theta(Z) \subset \mathcal{U}$ moves and deforms, and the loss function (in the form Eq.(12)) forces it to stay close to the points $u^{\eta_1}, \ldots, u^{\eta_N} \in G(\mathcal{A})$, hence approximately contains the whole set of possible solutions $G(\mathcal{A})$. Here is an ordinary differential equation (ODE, i.e., 1-D PDE) to visualize the pre-training and fine-tuning processes of MAD. We take the problem domain to be $\Omega = (-\pi, \pi) \subset \mathbb{R}$, the set of variable ODE parameters $\mathcal{A} = \mathbb{R}$, and the latent space $Z = \mathbb{R}$. For given

$\eta \in \mathcal{A}$, the ODE[4]

$$\begin{aligned} \frac{\mathrm{d}u}{\mathrm{d}x} &= \phi_\eta(x) = 2(x-\eta)\cos\big((x-\eta)^2\big), \\ u(-\pi) &= \sin\big((\pi+\eta)^2\big), \\ u(\pi) &= \sin\big((\pi-\eta)^2\big) \end{aligned}$$

has a unique solution

$$u^\eta(x) = \sin\big((x-\eta)^2\big),$$

and the corresponding physics-informed loss is

$$L^\eta[u] = \int_{-\pi}^{\pi} \big|u'(x) - \phi_\eta(x)\big|^2 \,\mathrm{d}x + \big|u(-\pi) - u^\eta(-\pi)\big|^2 + \big|u(\pi) - u^\eta(\pi)\big|^2.$$

In this problem, the neural network $f_\theta(x, z)$ has five hidden layers of width 64 and uses Sine activation function [28]. We sample 20 points equidistantly in the $[0, 2]$ interval as variable ODE parameters, and randomly select one η_{new} for the fine-tuning stage and the rest $\{\eta_i\}_{i\in\{1,\cdots,19\}}$ for the pre-training stage. Equidistant sampling is also used in spatial discretization $\mathcal{D}_{\text{r}}$ (i.e., $x_1 = -\pi < x_2 < \cdots < x_{128} = \pi$).

Pre-training In the pre-training stage, we utilizes the Adam optimizer to solve Eq.(13) with the regularization term omitted (i.e., taking $\sigma = \infty$) for simplicity. The pre-training iteration generates a sequence of $(\theta^{(m)}, \{z_i^{(m)}\}_{i\in\{1,\cdots,19\}})$, and terminates at $m = 200$ (enough for the neural network to learn the manifold) with the optimal $(\theta^*, \{z_i^*\}_{i\in\{1,\cdots,19\}})$. In order to visualize how the set $G_\theta(Z)$ evolves, we project the infinite-dimensional function space $\mathcal{U} = C([-\pi, \pi])$ onto a 2-dimensional plane as follows:

1. Given $u \in \mathcal{U}$, we compute its evaluations at the equidistant grid points to get a 128-dimensional vector
$$\mathrm{ev}(u) = [u(x_1), \ldots, u(x_{128})] \in \mathbb{R}^{128}.$$
2. Then the principal component analysis (PCA) method is used to project the 128-dimensional vector into the 2-dimensional plane.

Fig.4 visualizes how $G_\theta(Z)$ evolves and gradually fits $G(\mathcal{A})$ in pre-training stage.

Fine-tuning the Latent Vector Now assume that we have obtained a optimal network weight θ^* in the pre-training stage, and $G_{\theta^*}(Z)$ fits $G(\mathcal{A})$ well. This suggests that, for a given new ODE parameter $\eta_{\text{new}} \in \mathcal{A}$, there is some latent vector $z_{\text{new}} \in Z$ such that $G_{\theta^*}(z_{\text{new}}) \approx G(\eta_{\text{new}})$ holds. In the fine-tuning stage, we only optimize the latent vector z_{new} according to Eq.(14). As $z = z_{\text{new}}^{(m)}$ updates, the point $G_{\theta^*}(z_{\text{new}}^{(m)})$ moves on $G_{\theta^*}(Z)$, and finally converges to the approximate solution $G_{\theta^*}(z_{\text{new}}) \approx G(\eta_{\text{new}})$. The result is shown in Fig.5a, which conforms with what we expect in Fig.2.

Fine-tuning the Latent Vector with Model In the fine-tuning stage, we not only optimize $z = z_{\text{new}}^{(m)}$, but also fine-tune the network weight $\theta = \theta^{(m)}$ with $\theta^{(0)} = \theta^*$, as in Eq.(15). On the one hand, the point $G_{\theta^{(m)}}(z_{\text{new}}^{(m)})$ moves on the set $G_{\theta^{(m)}}(Z)$ as we update $z_{\text{new}}^{(m)}$. On the other hand, updating $\theta^{(m)}$ would make this set deform, which brings the point $G_{\theta^{(m)}}(z_{\text{new}}^{(m)})$ closer to the true solution $G(\eta_{\text{new}})$, as already illustrated in Fig.3. The result is shown in Fig.5b.

[4]Actually, setting one of $u(-\pi)$ and $u(\pi)$ is already enough to determine the solution. We utilize such a superfluous boundary condition to make training easier.

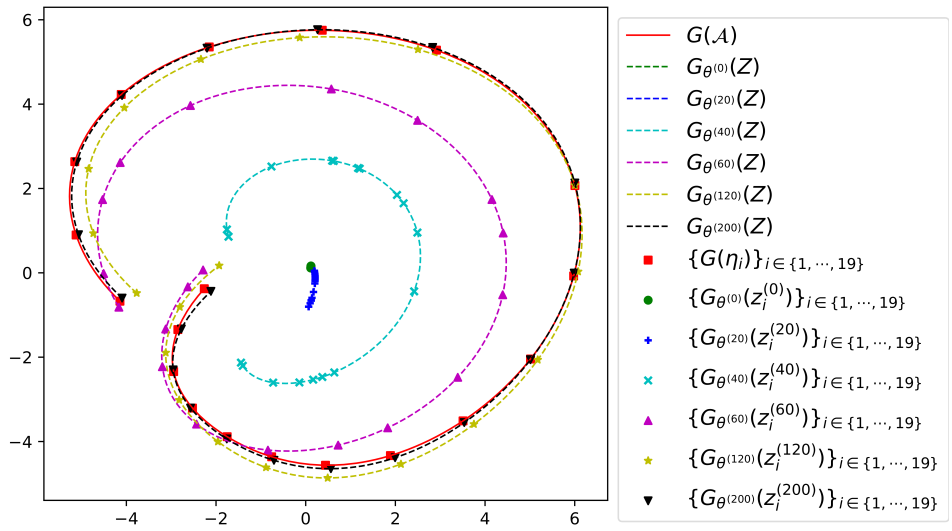
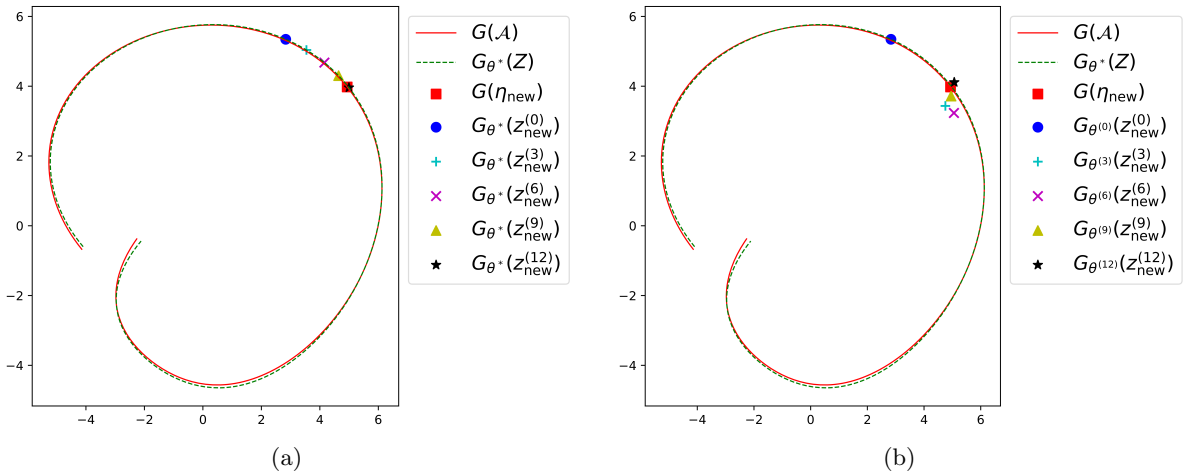


Figure 4: Visualization of the Meta-Auto-Decoder pre-training process in the ODE problem. The infinite-dimensional function space $\mathcal{U}$ is projected onto a 2-dimensional plane. The set of possible solutions forms a 1-dimensional manifold (i.e. curve) $G(\mathcal{A})$, and the marked points correspond to those ODE parameters $\{\eta_i\}_{i\in\{1,\cdots,19\}}$ used for pre-training. For give θ, our Meta-Auto-Decoder $f_\theta(\cdot, z)$ represents curves $G_\theta^{(m)}(Z)$ parametrized by $z \in Z$. As the number of iteration m increases, the network weight $\theta = \theta^{(m)}$ updates, making curves evolve and finally fit the target manifold $G(\mathcal{A})$. The points $G_{\theta^{(m)}}(z_i^{(m)}) = f_{\theta^{(m)}}(\cdot, z_i^{(m)})$ are also marked on the curves.

4 Numerical Experiments

To test the effectiveness of the MAD method, we apply it to solve three different problems: (1) Burgers' equation with variable initial conditions; (2) Laplace's equation with variable solution domains and boundary conditions; (3) Maxwell's equations with variable equation coefficients. In these problems, the PDE parameters vary from different aspects, including initial conditions, domain shapes and coefficients. For each experiment, we divide the PDE parameters into two sets, S_1 and S_2, of which S_1 is used in the pre-training stage and S_2 is used in the fine-tuning stage for evaluation. For ease of expression, we declare the following abbreviations:

- *From-Scratch*: Train the model from scratch based on the PINNs method for all PDE parameters in S_2, case-by-case.
- *Transfer-Learning* [12]: Randomly select a PDE parameter in S_1 for pre-training stage based on the PINNs method, and then load the obtained weight in pre-training stage for PDE parameters in S_2 during fine-tuning stage.
- *MAML* [22, 23]: Meta-train the model for all PDE parameters in S_1 based MAML method. In the meta-testing stage, we load the pre-trained weight θ^* and fine-tune the model for each PDE parameter in S_2.
- *Reptile* [24]: Similar to *MAML*, except that the model weight is updated in the meta-training stage using the Reptile method.
- *MAD-L*: Pre-train the model for all PDE parameters in S_1 based on our proposed method and then load and freeze the pre-trained weight θ^* for the second stage. In the fine-tuning stage, we choose a z_i^* obtained in the pre-training stage to initialize a latent vector for each PDE parameter in S_2, and fine-tune the latent vector. The selection of z_i^* is based on the distance between η_{new} and η_i.
- *MAD-LM*: Different from *MAD-L* that freezes the pre-trained weight, we fine-tune the model weight θ and the latent vector z simultaneously in the fine-tuning stage.



(a) (b)

Figure 5: Visualization of the Meta-Auto-Decoder fine-tuning process in the ODE problem, still projected onto a two-dimensional plane. (a) As we update $z = z_{\text{new}}^{(m)}$ according to the loss function Eq.(14), the corresponding point $G_{\theta^*}(z_{\text{new}}^{(m)})$ moves on the curve $G_{\theta^*}(Z)$, and finally converges to a point close to the true PDE solution $G(\eta_{\text{new}})$. (b) If we apply Eq.(15) instead, and fine-tune the network weight $\theta = \theta^{(m)}$ simultaneously (with initialization $\theta^{(0)} = \theta^*$), the point $G_{\theta^{(m)}}(z_{\text{new}}^{(m)})$ would also converge to $G(\eta_{\text{new}})$.

Unless otherwise specified, the following default configurations are used for the experiments:

- In each iteration, the batch sizes are selected as $(M_r,\ M_{bc}) = (8192,\ 1024)$.
- For fairness of comparison, the network architectures of all methods involved in comparison are the same except for the input layer due to the latent vector. For Burgers' equation and Laplace's equation, the standard fully-connected neural networks with 7 fully-connected layers and 128 neurons per hidden layer are taken as a default network structure. For Maxwell's equations, the MS-SIREN network architecture [29] is used that has 4 subnets, each subnet has 7 fully-connected layers and 64 neurons per layer. It is worth noting that our proposed method has gains in different network architectures, and we choose the default network architecture that can achieve high accuracy for the *From-Scratch* method to conduct our comparative experiments.
- Sine function [28] is used as the activation function, as it exhibits better performance than other alternatives such as ReLU and Tanh.
- The dimension of latent vector z is set to 128 for Burgers' equation and Laplace's equation, and 16 for Maxwell's equations.
- The Adam optimizer [30] is used with the initial learning rate set to 1e-3 or 1e-4 (whoever achieves the best performance). When the training process reaches 40%, 60% and 80%, the learning rate is attenuated by half.

To evaluate the accuracy of the model, we compare the predicted results with the reference solutions to obtain L_2 *error*. Since $|S_2|$ PDE parameters are tested in the fune-tuning stage, we provide the mean and the 95% confidence interval of L_2 *error*s. Unless otherwise specified, all the experiments are conducted under the MindSpore[5] and trained on the Ascend 910 AI processors[6]. Readers can read our source code[7] for implementation details.

[5]https://www.mindspore.cn/
[6]https://e.huawei.com/en/products/servers/ascend
[7]https://gitee.com/mindspore/mindscience/tree/master/MindElec/

4.1 Burgers' Equation

We consider the 1-D Burgers' equations:

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} &= \nu \frac{\partial^2 u}{\partial x^2}, \quad x \in (0,1), t \in (0,1], \\ u(x,0) &= u_0(x), \quad x \in (0,1). \end{aligned} \tag{16}$$

Eq.(16) can model one-dimensional flow of a viscous fluid, where u is the velocity, ν is the viscosity coefficient and the initial condition $u_0(x)$ is the changing parameter of the PDEs, i.e. $\eta = u_0(x)$. The initial condition $u_0(x)$ is generated using Gaussian random field (GRF) [31] according to $u_0(x) \sim \mu = \mathcal{N}(0; 1000(-\Delta + 9I)^{-3})$ with periodic boundary conditions. We set the viscosity to $\nu = 0.01$ and solve the Eq.(16) using open source code implemented by [8] to generate the reference solutions. The spatiotemporal mesh size of the ground truth is 1024×101. In order to solve the Eq.(16) better by using the PINNs-based method, we use the hard constraint on periodic boundary condition mentioned in [32].

Unlike FNO [8] which learns the operator mapping from the initial condition $u(\cdot, 0)$ to the solution at time one $u(\cdot, 1)$, our method learns the operator mapping from the initial condition $u(\cdot, 0)$ to the solution in the entire duration $u(\cdot, \cdot)$, which is a more challenging task. In addition, our approach uses only the information inherent in the PDEs and does not require any supervised data, whereas FNO is data-driven and requires a large amount of labeled data.

We generated 150 different initializations of $u_0(x)$ using GRF and randomly selected 100 cases as S_1. The remaining 50 scenarios are used as S_2 for fine-tuning. For *MAD-L* and *MAD-LM*, the pre-training stages run for 50k iterations while the *Transfer-Learning* pre-trains 3k steps since it only handles one single case.

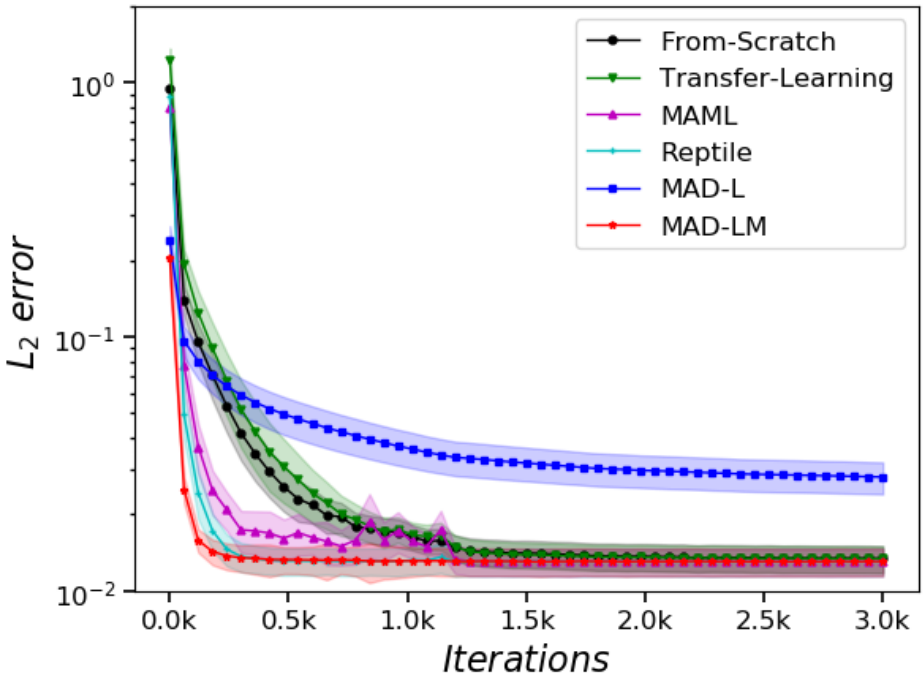


Figure 6: **Burgers' equation:** The mean L_2 *error* convergence with respect to the number of training iterations under different method.

Fig.6 shows that all curves can converge to almost the same accuracy (L_2 *error* is all close to 0.013), except for *MAD-L*. The shaded part of each curve in Fig.6 is the 0.95 confidence interval which reflects the variations among these $|S_2|$ L_2 *error*s. The range of the shaded part corresponding to each curve in Fig.6 is small, which indicates that most of L_2 *error*s are on the same level. Fig.6 shows that *MAD-LM* is the fastest in terms of the convergence speed. In addition, we can see that the *MAML*, *Reptile* and *MAD-LM* roughly converge after 200 iterations, while the *From-Scratch* and

Transfer-Learning require about 1200 iterations to converge. The *MAD-L* cannot converge to the ideal accuracy which probably results from the c-gap introduced in Sec.3.4. The pre-trained model cannot fit the manifold accurately and the revelation is that we need to update the model during fine-tuning stage as described by Sec.3.4. With the model weight and latent vector updated together, the *MAD-LM* obtains similar L_2 *error* and only requires 17% of the training iterations compared with *From-Scratch*. In this example, *Transfer-Learning* does not achieve any faster convergence than *From-Scratch*, and is even slightly slower than *From-Scratch* in the early stage of training. This means the pre-training stage fails to obtain an initialization that favors fine-tuning for most cases in S_2.

Fig.7 shows model predictions of *MAD-L* and *MAD-LM* compared with the reference solutions under a randomly selected $u_0(x)$ in S_2. The predictions of *MAD-L* are in overall approximate agreement with the reference solutions, but the fit is poor at the spikes and troughs. However, the prediction results of *MAD-LM* is almost the same as that of the reference solutions.

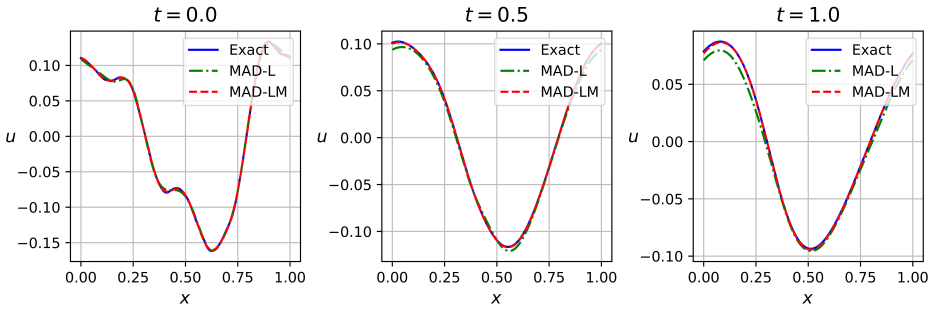


Figure 7: **Burgers' equation:** Reference solutions vs. model predictions at $t = 0.0$, $t = 0.5$ and $t = 1.0$, respectively.

We investigated the effect of the number of samples $|S_1|$ in the pre-training stage on *MAD-L* and *MAD-LM*. Fig.8 shows that the accuracy of *MAD-L* after convergence increases with $|S_1|$. However, when $|S_1|$ reaches about 200, increasing $|S_1|$ does not improve the accuracy of the *MAD-L*. This result is consistent with the phenomenon shown in Fig.3. More samples in the pre-training stage allow the $G_{\theta^*}(Z)$ to gradually fall within the region formed by $G(A)$. However, when $|S_1|$ reaches a certain level, the $G_{\theta^*}(Z)$ only swings in the region of $G(A)$. Only optimizing z can make the solution move inside the manifold formed by $G_{\theta^*}(Z)$, but $u^{\eta_{\text{new}}}$ may not be close enough to the manifold. Therefore, in order to obtain a more accurate solution, we need to fine-tune z and θ simultaneously. Fig.9 shows that the accuracy and convergence speed of *MAD-LM* do not change significantly with the increase of samples in the pre-training phase. It is only when the number of samples is very small (i.e., $|S_1| = 10$) that the early convergence speed is significantly affected. This shows that *MAD-LM* can perform well in the fine-tuning stage without requiring a large number of samples during the pre-training stage.

4.2 Laplace's Equation

We consider the 2-D Laplace's equation as follows:

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= 0, \qquad (x, y) \in \Omega, \\ u(x, y) &= g(x, y), \qquad (x, y) \in \partial\Omega. \end{aligned} \tag{17}$$

where the shape of Ω and the boundary condition $g(x, y)$ are the variable parameters of the PDEs, i.e. $\eta = (\Omega, g(x, y))$. In this example, we test the performance of MAD by varying the shape of the triangular domain Ω. We randomly choose three points from the circumference of the unit circle as the three vertices of the triangles. Given that h is the boundary condition on the unit circle, we use GRF to generate $h \sim \mathcal{N}(0, 10^{3/2}(-\Delta + 100I)^{-3})$ with periodic boundary conditions. The analytical solution of the Laplace's equation on the unit circle can be obtained by using the Fourier method [33]. Then, we use the analytical solution on the three sides of the triangle as the boundary condition $g(x, y)$. We generated 150 samples and randomly selected 100 samples as S_1, the remaining 50 cases are used as S_2. It should be declared that each sample corresponds to a different h inside the boundary circle.

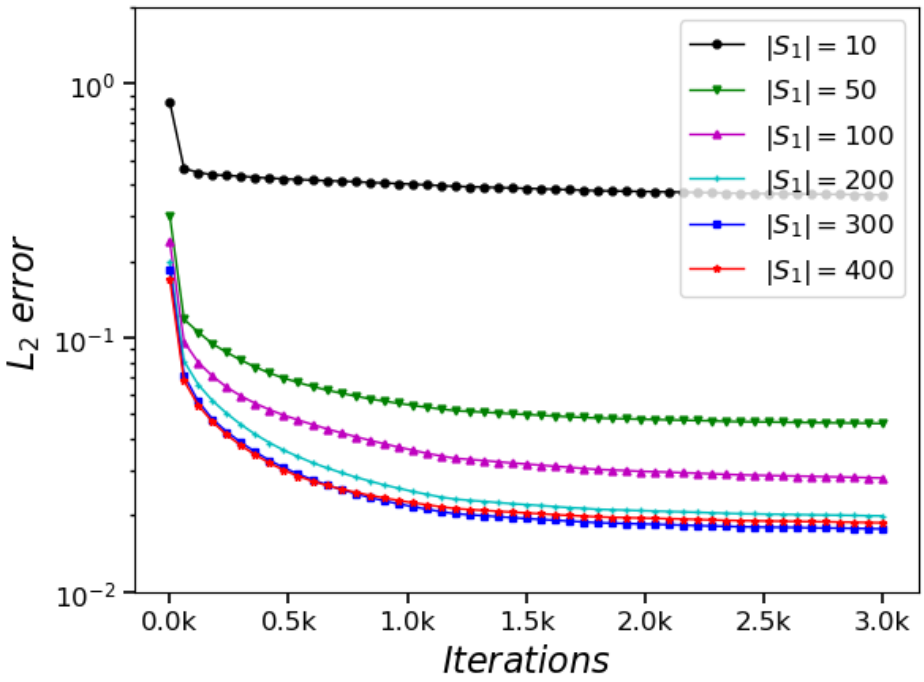


Figure 8: **Burgers' equation:** The mean L_2 *error* of *MAD-L* convergence with respect to the number of training iterations under different numbers of samples in S_1.

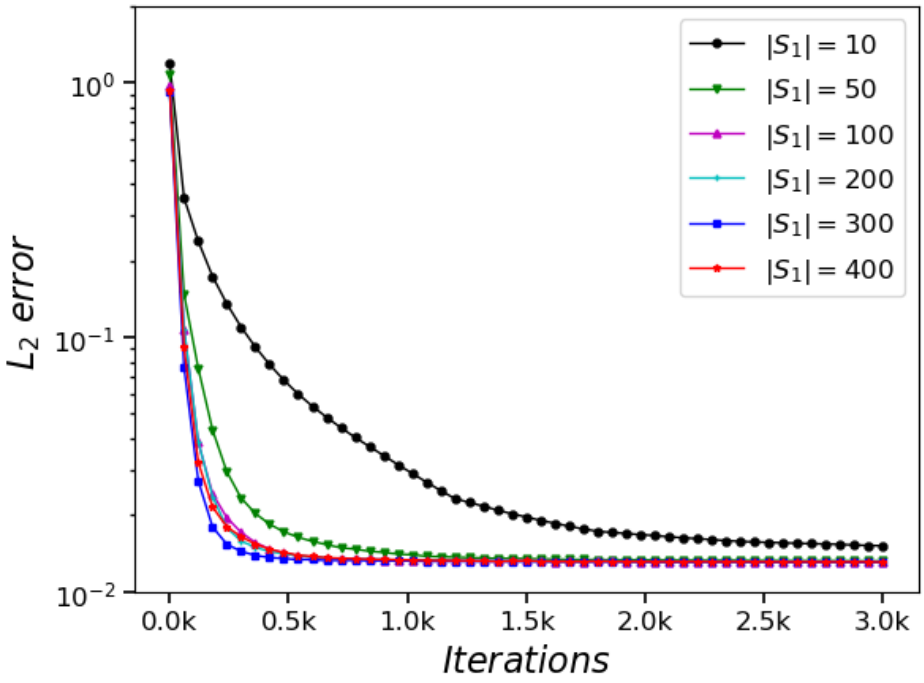


Figure 9: **Burgers' equation:** The mean L_2 *error* of *MAD-LM* convergence with respect to the number of training iterations under different numbers of samples in S_1.

For each sample, we randomly select 16×1024 points from the interior of the triangle and obtain the analytic solutions corresponding to these points to evaluate the accuracy of the model. When we apply the MAD method to solve this problem, it is not convenient to measure the distance between η_{new} and η_i since the variable PDE parameter η is the shape and boundary condition of the solution domain. Therefore, the average of $|S_1|$ latent vectors obtained in the pre-training stage is used as the initialization of z in the fine-tuning stage.

For the training of *From-Scratch*, the pre-training and fine-tuning of *Transfer-Learning*, and the fine-tuning of *MAD-L* and *MAD-LM*, we set the total number of iterations to 10k. For the pre-training of *MAD-L* and *MAD-LM*, we set the total iterations to 50k.

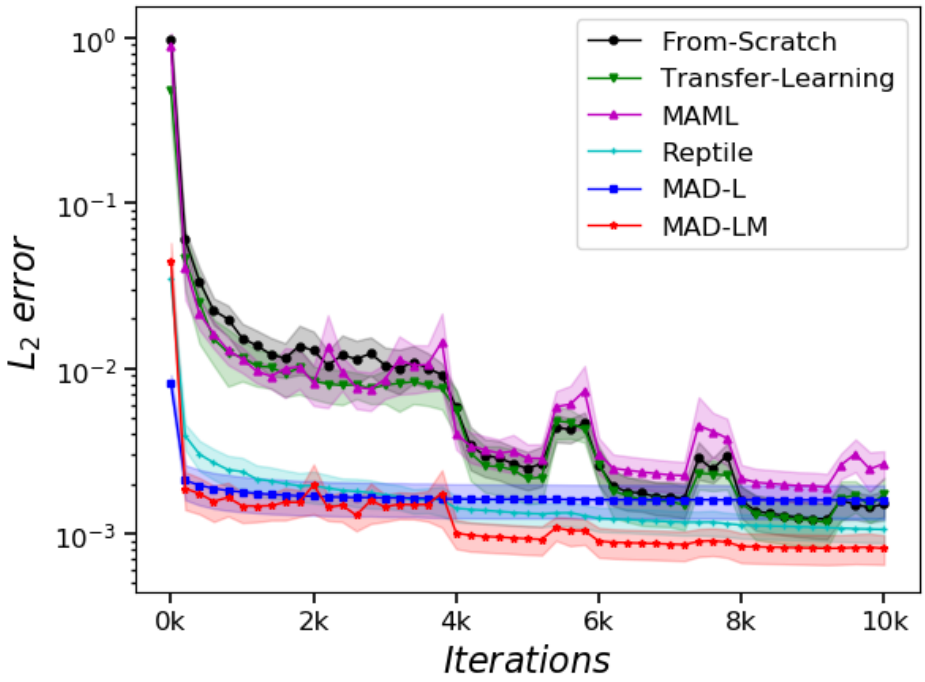


Figure 10: **Laplace's equation:** The mean L_2 *error* convergence with respect to the number of training iterations under different method.

Fig.10 shows that all curves can converge to almost the same accuracy (L_2 *error* is all close to 0.001). Both *MAD-L* and *MAD-LM* converges much faster than other methods. Unsurprisingly, the final L_2 *error* of *MAD-L* is slightly higher than that of *MAD-L* since *MAD-L* only optimizes on the latent vector z. In this case, the information learned by *Transfer-Learning* and *MAML* in the first stage does not seem to work for the second stage, so they do not exhibit significantly faster convergence in the second stage. *Reptile* also shows a generalization capability similar to our method in this example.

Fig.11 shows the predictions of *MAD-L* and *MAD-LM* compared with the analytical solutions under a randomly selected sample in S_2. To the naked eye, the prediction results of *MAD-L* and *MAD-LM* are almost identical to those of the analytical solution. However, the L_2 *error* of *MAD-LM* is 0.0015, and that of *MAD-L* is 0.0005.

4.3 Time-Domain Maxwell's Equations

We consider the time-domain 2-D Maxwell's equations with a point source in the transverse Electric (TE) mode [34]:

$$\begin{aligned}
\frac{\partial E_x}{\partial t} &= \frac{1}{\epsilon_0 \epsilon_r} \frac{\partial H_z}{\partial y}, \\
\frac{\partial E_y}{\partial t} &= -\frac{1}{\epsilon_0 \epsilon_r} \frac{\partial H_z}{\partial x}, \\
\frac{\partial H_z}{\partial t} &= -\frac{1}{\mu_0 \mu_r} \left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} + J \right).
\end{aligned} \tag{18}$$

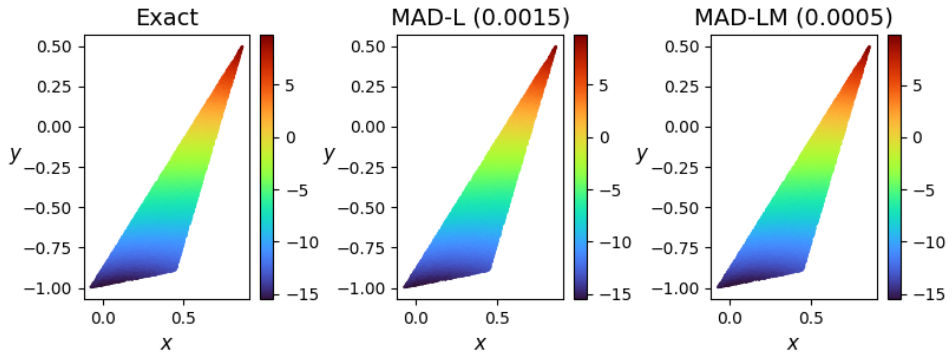


Figure 11: **Laplace's equation:** Analytical solutions, model predictions of *MAD-L* and *MAD-LM* (left to right).

where E_x and E_y are the electric field in the Cartesian coordinate along x and y axis, respectively. H_z is the magnetic field perpendicular to the horizontal plane. The equation coefficients ϵ_0 and μ_0 are the permittivity and permeability in vacuum, respectively. The equation coefficients ϵ_r and μ_r are the relative permittivity and relative permeability of the media, respectively. [29] uses the modified PINNs method to solve the Eq.(18) with fixed $\epsilon_r = 1$ and $\mu_r = 1$. However, in this paper, $(\epsilon_r,\ \mu_r)$ are the variable parameters of the PDEs, i.e., $\eta = (\epsilon_r,\ \mu_r)$, which corresponds to the media properties in the simulation region. J represents a known source function and we set it to a Gaussian pulse. In temporal, this function can be expressed as:

$$J(x, y, t) = e^{-(\frac{t-d}{\tau})^2}\delta(x - x_0)\delta(y - y_0). \tag{19}$$

where d is the temporal delay, τ is a pulse-width parameter, $\delta(\cdot)$ is the Dirac function used to represent the point source, and $(x_0, y_0) = (0.5, 0.5)$ is the location of the point source, $\tau = 3.65 \times \sqrt{2.3}/(\pi f)$ and the characteristic frequency f is set to be $1GHz$. The initial electromagnetic field is zero everywhere and the boundary condition is the stranded Mur's second-order absorbing boundary condition. We solve the problem above on $\Omega = [0, 1]^2 \times [0, 4e - 9]$. To solve the singularity problem caused by the point source, we use the $\delta(\cdot)$ function approximation method and the lower bound uncertainty weighting method proposed by [29]. To measure the accuracy of the model, the reference solution is obtained through the finite-difference time-domain (FDTD) [35] method with the spatio-temporal resolutions $(\Delta_x, \Delta_y, \Delta_t) = (0.005, 0.005, 1e - 11)$.

We collect 25 pairs of $(\epsilon_r,\ \mu_r)$ at equal intervals from the region of $[1, 5]^2$ and randomly select 20 samples as S_1 with the rest 5 samples as S_2. For the training of *From-Scratch*, the pre-training and fine-tuning of *Transfer-Learning*, we set the total number of iterations to 100k. For the pre-training of *MAD-L* and *MAD-LM*, we set the total iterations to 200k. For the fine-tuning of *MAD-L* and *MAD-LM*, we set the total number of iterations to 50k.

Fig.12 shows that all methods converge to approximate accuracy (L_2 *error* close to 0.05), except for *MAD-L*. However, the search on the low-dimensional space only brings significantly gains in terms of convergence speed for *MAD-L* as its curve flattens at about $1.0K$ iterations (L_2 *error* close to 0.092). *MAD-LM* can also converge faster than *From-Scratch*, *Transfer-Learning* and *Reptile* while maintaining high accuracy. The *MAML* fails to converge during the pre-training stage since it involves the computation of second-order information and the Dirac-delta singular function brings optimization difficulty. *Reptile* does not show good task generalization, probably due to the inherent singularity of this problem.

The instantaneous electromagnetic fields at time $4ns$ of *MAD-L* and *MAD-LM* compared with the reference FDTD results when $(\epsilon_r,\ \mu_r) = (3, 5)$ are depicted in Fig.13a and Fig.13b, respectively. By observing the absolute error between the model predictions and the reference FDTD results, we can see that *MAD-LM* can achieve a lower absolute error. Specifically, the L_2 *error* of *MAD-L* is 0.101 and that of *MAD-LM* is 0.038 in this scenario.

To sum up, the proposed MAD method converges faster than other deep learning approaches for solving parametric PDEs while maintaining high accuracy. Compared with solving a single PDE using the PINNs method, our approaches can improve the convergence speed by 5-50x. Compared with other

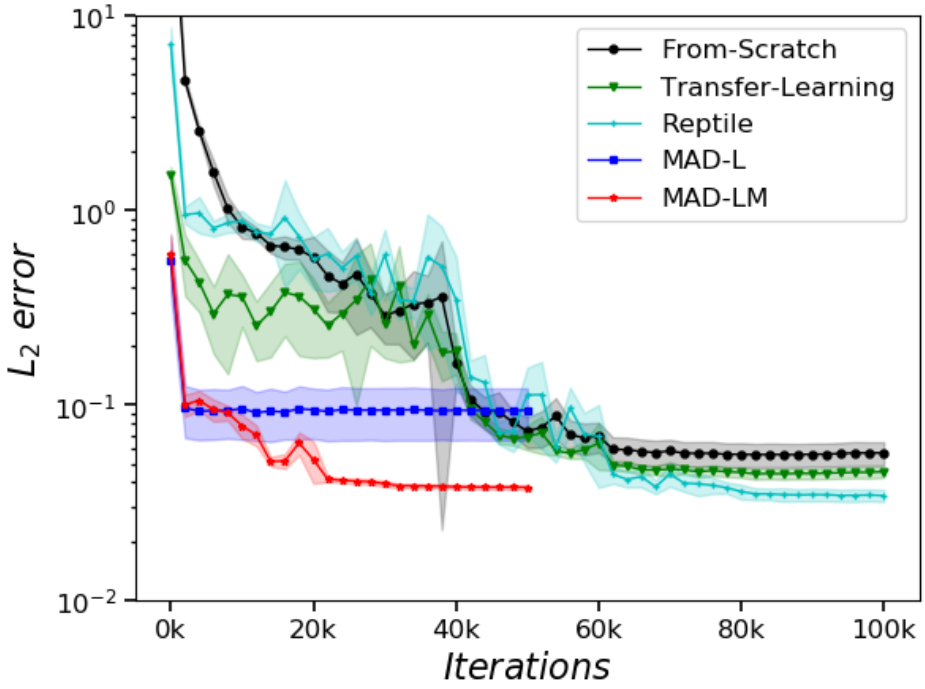


Figure 12: **Maxwell's equations:** The mean L_2 *error* convergence with respect to the number of training iterations under different method.

transfer/meta learning approaches, MAD exhibits a more stable speed-up performance especially for difficult tasks.

5 Conclusions

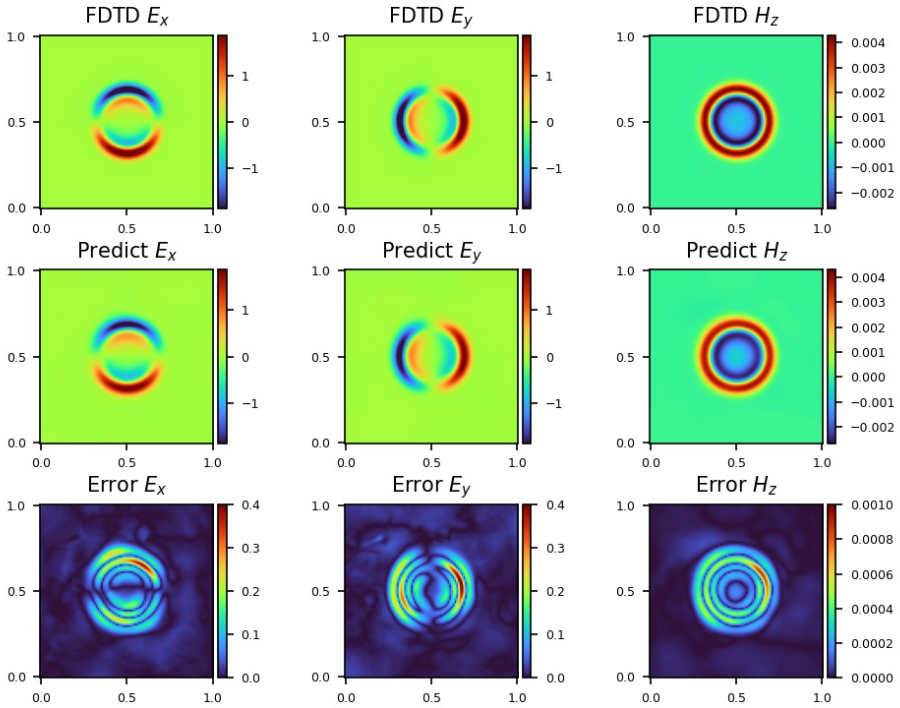
In this paper, we propose MAD, a novel unsupervised method for solving parametric PDEs. MAD solves parametric PDEs from a meta-learning perspective and implicitly encodes each PDE parameter as a latent vector z based on the idea of auto-decoder. MAD encodes the information of governing equations and boundary conditions into neural network by optimizing the physics-informed losses. The fine-tuning process of latent vector z is essentially the back propagation process of physics-informed losses.

From the perspective of manifold learning, the neural network $G_{\theta^*}(Z)$ learned during the pre-training stage provides a good estimation of the exact solution manifold $G(\mathcal{A})$. The fine-tuning stage is to search the solution $f_{\theta^*}(\cdot, z_i)$ on the manifold $G_{\theta^*}(\cdot)$ via fine-tune latent vector z. In some cases, the exact solution does not lie in the learned manifold from the pre-training stage, so it is not enough to fine-tune the latent vector z only in the fine-tuning stage, due to the existence of the c-gap. We need to fine-tune the model weight θ based on the pre-trained model θ^* together with the latent vector z.

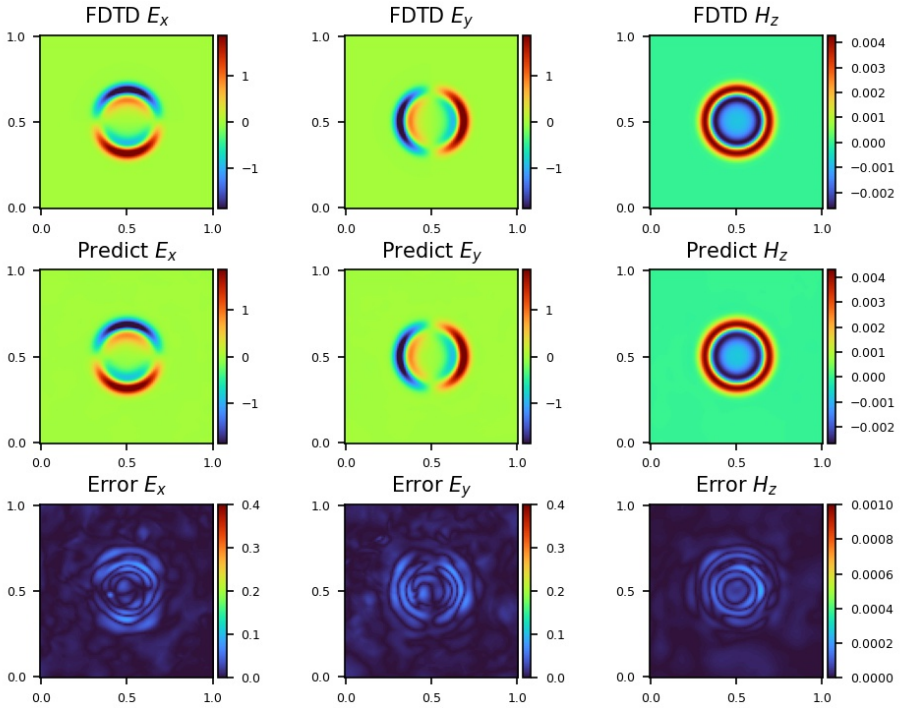
Through extensive numerical experiments, we prove that the MAD method can significantly improve the convergence speed of fine-tuning compared to other deep learning method.

For future works, we plan to explore the following promising directions:

- A rigorous mathematical theory of the MAD method needs to be developed to explain its attractive and robust performance.
- More neural network structures other than a single latent vector a can be further explored to encode the latent space.
- The applications of MAD can be naturally extended to other meta-learning [21] fields.



(a) $MAD - L$



(b) $MAD - LM$

Figure 13: **Maxwell's equations:** Model predictions of *MAD-L* (a) and *MAD-LM* (b) vs. the FDTD solutions at $t = 4ns$. **Top:** The numerical results of (E_x, E_y, H_z) by FDTD method; **Middle:** The predicted (E_x, E_y, H_z) through the deep learning model; **Bottom:** The absolute error between model predictions and the reference solutions.

References

[1] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[2] Albert Cohen and Ronald DeVore. Approximation of high-dimensional parametric pdes. *Acta Numerica*, 24:1–159, 2015.

[3] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

[4] Olgierd Cecil Zienkiewicz, Robert Leroy Taylor, Perumal Nithiarasu, and JZ Zhu. *The finite element method*, volume 3. McGraw-hill London, 1977.

[5] Tadeusz Liszka and Janusz Orkisz. The finite difference method at arbitrary irregular grids and its application in applied mechanics. *Computers & Structures*, 11(1-2):83–95, 1980.

[6] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[7] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *arXiv preprint arXiv:2005.03180*, 2020.

[8] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[9] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[10] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

[11] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[12] E Weinan and Bing Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[13] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020.

[14] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016.

[15] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[16] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

[17] Shuhao Cao. Choose a transformer: Fourier or galerkin. *arXiv preprint arXiv:2105.14995*, 2021.

[18] Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[19] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *arXiv preprint arXiv:2006.09535*, 2020.

[20] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6):060801, 2021.

[21] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[22] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[23] Antreas Antoniou, Harri Edwards, and Amos Storkey. How to train your maml. In *Seventh International Conference on Learning Representations*, 2019.

[24] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2(3):4, 2018.

[25] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7343–7353, 2018.

[26] Vincent Sitzmann, Eric R. Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. In *Proc. NeurIPS*, 2020.

[27] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[28] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[29] Xiang Huang, Hongsheng Liu, Beiji Shi, Zidong Wang, Kang Yang, Yang Li, Bingya Weng, Min Wang, Haotian Chu, Jing Zhou, Fan Yu, Bei Hua, Lei Chen, and Bin Dong. Solving partial differential equations with point source based on physics-informed neural networks, 2021.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Yang Liu, Jingfa Li, Shuyu Sun, and Bo Yu. Advances in gaussian random field generation: a review. *Computational Geosciences*, 23(5):1011–1047, 2019.

[32] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *arXiv preprint arXiv:2102.04626*, 2021.

[33] Dirichlet Problem in the Circle and the Poisson Kernel, 7 2021. [Online; accessed 2021-09-13].

[34] Stephen D Gedney. Introduction to the finite-difference time-domain (fdtd) method for electromagnetics. *Synthesis Lectures on Computational Electromagnetics*, 6(1):1–250, 2011.

[35] John B Schneider. Understanding the finite-difference time-domain method. *School of electrical engineering and computer science Washington State University*, 28, 2010.