Proceedings of Machine Learning Research vol 107:269–286, 2020

Deep Domain Decomposition Method: Elliptic Problems

Wuyang Li LIWY648@NENU.EDU.CN
Qian Xuesen Laboratory of Space Technology, China Academy of Space Technology, Beijing 100194, China.
School of Mathematics and Statistics, Northeast Normal University, Changchun 130012, China.

Xueshuang Xiang* XIANGXUESHUANG@QXSLAB.CN
Qian Xuesen Laboratory of Space Technology, China Academy of Space Technology, Beijing 100194, China.

Yingxiang Xu YXXU@NENU.EDU.CN
School of Mathematics and Statistics, Northeast Normal University, Changchun 130012, China.

Abstract

This paper proposes a deep-learning-based domain decomposition method (DeepDDM), which leverages deep neural networks (DNN) to discretize the subproblems divided by domain decomposition methods (DDM) for solving partial differential equations (PDE). Using DNN to solve PDE is a physics-informed learning problem with the objective involving two terms, domain term and boundary term, which respectively make the desired solution satisfy the PDE and corresponding boundary conditions. DeepDDM will exchange the subproblem information across the interface in DDM by adjusting the boundary term for solving each subproblem by DNN. Benefiting from the simple implementation and mesh-free strategy of using DNN for PDE, DeepDDM will simplify the implementation of DDM and make DDM more flexible for complex PDE, e.g., those with complex interfaces in the computational domain. This paper will firstly investigate the performance of using DeepDDM for elliptic problems, including a model problem and an interface problem. The numerical examples demonstrate that DeepDDM exhibits behaviors consistent with conventional DDM: the number of iterations by DeepDDM is independent of network architecture and decreases with increasing overlapping size. The performance of DeepDDM on elliptic problems will encourage us to further investigate its performance for other kinds of PDE and may provide new insights for improving the PDE solver by deep learning.

Keywords: Domain decomposition methods, Deep learning, Interface problems.

1. Introduction

In the realms of engineering and scientific computing, the domain decomposition methods (DDM) has attracted a great deal of interest and is well-known as an efficient approach for solving partial differential equations (PDE). The main idea of DDM is first splitting the computational domain into smaller subdomains and then solving in parallel the subproblems defined in subdomains, along with exchanging solution information between adjacent subdomains. When combined with the discretization of PDE by finite element methods (FEM) or finite difference methods (FDM), DDM can achieve remarkable performance. This paper will propose an approach named DeepDDM which investigates the behavior of discretizing PDE by deep neural networks (DNN) in DDM. Compared with DDM-FEM/DDM-FDM, DeepDDM will simplify the implementation of DDM. Especially, DeepDDM is more flexible for complex PDE, e.g., those with complex interfaces in the computational domain.

. *Corresponding author.

Due to the success of deep learning (DL) in engineering, the research using deep learning to solve PDE has provoked interest from numerous researchers. Recently, many relevant papers have been published focusing on different topics. Raissi et al. (2019) presents physics-informed neural networks (PINN) that take initial conditions and boundary conditions as the penalties of the optimization objective loss function. By using the backpropagation algorithm and automatic differentiation, it realizes the calculation of the high-order differential in the framework of TensorFlow. The network can achieve good accuracy for both forward and inverse problems. Sirignano and Spiliopoulos (2018) proposes a deep Galerkin method, which is similar to PINN in that initial conditions and boundary conditions are also added as penalty terms into the optimization objective loss function. The difference is that Sirignano and Spiliopoulos (2018) bypasses the high-order differentiation of the neural network through the Monte Carlo method. For well-known reasons, the computational cost of high-order differentiation of neural networks is exorbitant. Another method, a deep Ritz method, is presented by E et al. (2017), in which different network architectures and loss functions are designed for variational problems, particularly the ones that arise from PDE. The large multiple layer convolutional neural network is used to solve and discover evolutionary PDE by Long et al. (2019). Recently, a large number of researchers have attempted to use deep learning tools to solve various practical problems Rudd et al. (2013); Berg and Nyström (2018).

In the field of combining machine learning and domain decomposition, Mai-Duy and Tran-Cong (2002) presented a combination of radial basis function network methods and domain decomposition technique for approximating functions and solving Poisson equations. Recently, as we prepared for this paper, some related works of combining domain decomposition method and deep learning have appeared on the Internet. Li et al. (2019) presented D3M, a method that combines Deep Ritz method and DDM to solve general PDE in parallel. However, our approach focus on the performance of combing PINN and DDM for Elliptic problems, especially for the problem with a complex interface. Dwivedi et al. (2019) presented a Distributed PINN that divides the problem into many subproblems according to a domain splitting, and then use the total loss, i.e. the sum of all subproblem's loss and the loss on the interface, to train all the subproblems at the same time. We notice that the training objective of each subproblem in Distributed PINN is related to its neighbors at each training step. That means Distributed PINN is not DDM-style method, since we need to first independently solve each subproblem and then exchange the interface information in DDM, like D3M and the proposed DeepDDM. On the application front, Kissas et al. (2020) applied PINN in different domains of cardiovascular flows modeling with the similar spirit of Distributed PINN.

We present a deep-learning-based domain decomposition framework, called the deep domain decomposition method (DeepDDM), which extracts the spirit of deep learning and domain decomposition. Using DNN to solve PDE is a physics-informed learning problem with the objective involving two terms, domain term and boundary term, which respectively make the desired solution satisfy the PDE and corresponding boundary conditions Raissi et al. (2019). By dividing the domain of interest into many subdomains, DeepDDM alternatively solves each subproblem by DNN and exchanges interface information until convergence. The decomposition of the domain of interest is based on the properties of the original physical background or the convenience of computing. Since DNN makes solving PDE a simple learning problem and is actually a mesh-free strategy, DeepDDM will simplify the implementation of DDM and make DDM more flexible for complex PDE.

This paper will first investigate the performance of DeepDDM on elliptic problems, including a model problem and an interface problem. For the model problem with the Dirichlet boundary con-

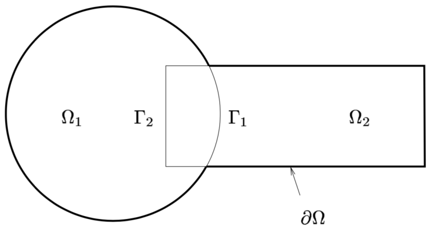


Figure 1: The diagram used by Schwarz in 1870.

dition, we consider the original problem divided into two or four subproblems in coordination with overlapping information and varying discrete functional spaces (network architectures). The results demonstrate that DeepDDM has properties similar to DDM with FEM or FDM: the necessary number of iterations is independent of the network architecture, given the number of subproblems; the necessary number of iterations increases along with the number of subproblems; the necessary number of iterations decreases according to increasing overlap size; and the numerical convergence rate coincides with the analytic convergence rate in various cases. Next, we investigate the performance of DeepDDM on an artificial interface problem, which is divided into two subproblems. Although only the elementary Dirichlet-Neumann interface condition is adopted, DeepDDM also exhibits good performance for the interface problem, just as for the model problem. In addition, DeepDDM reaches the relative error accuracy of 10^{-3} within several iterations for varying network architectures, especially for the case that the coefficient contrast (the ratio of the maximal diffusion coefficient on the minimal diffusion coefficient of the interface problem) is 20. In conclusion, as an approach with simple implementation, DeepDDM produces similar results as DDM with FEM or FDM and may improve the performance of using DDM for interface problems.

This paper is structured as follows. In section 2 and section 3, we provide a brief review of DDM and the PINN, respectively. A detailed description of DeepDDM for the general differential operator and boundary condition is presented in section 4. We implement and test the performance of DeepDDM on two kinds of elliptic problems in section 5. The section 6 will conclude this paper and present some directions for future work.

2. Domain decomposition methods

DDM are parallel, potentially fast, robust algorithms for solving PDE, which has been discretized using, e.g., FEM or FDM. Even in the 19th century, Hermann Schwarz considered a Poisson problem set on a union of simple geometries and introduced an alternating iterative method in Schwarz (1870). Nearly a century later, Pierre-Louis Lions presented parallel Schwarz methods for parallel computing in Lions (1988) after the parallel computing capability became available. For the discrete system, the multiplicative Schwarz method, sequential algorithms in nature, is introduced in Chan and Mathew (1994); Smith et al. (2004) and the additive Schwarz method, parallel algorithms, is studied by Dryja and Widlund in Dryja and Widlund (1990). With the wide use of parallel computers, DDM develop rapidly and provide many fast iterative algorithms and efficient preconditioners. Balancing domain decomposition by constraints (BDDC) and finite element tearing and interconnecting (FETI) have been researched for a variety of PDE as typical representatives of primal methods and dual methods, respectively Mathew (2008); Dolean et al. (2015); Lanser (2015); Pechstein and Dohrmann (2017). Moreover, a hybrid method between primal and dual, dual-primal finite element tearing and interconnecting (FETI-DP), enforcing equality of the solution at the subdomain interface by Lagrange multipliers, except at the subdomain corner, was initially introduced by Farhat et al. (2001) and has developed into many varieties Klawonn et al. (2014); Dolean et al. (2015).

Here we briefly introduce a classical DDM, the Schwarz alternating method of Schwarz in 1870, as follows,

$$\begin{aligned} \Delta u &= 0, \quad \text{in}\, \Omega, \\ u &= g, \quad \text{on}\, \partial\Omega, \end{aligned} \tag{1}$$

where Ω, shown as Figure 1, may be divided into a disk Ω_1 and a rectangle Ω_2, $\Omega = \Omega_1 \cup \Omega_2$, with interfaces $\Gamma_1 := \partial\Omega_1 \backslash \partial\Omega$ and $\Gamma_2 := \partial\Omega_2 \backslash \partial\Omega$, and g is a function given as the boundary condition. The Schwarz alternating method starts with an initial guess u_2^0 along Γ_1 and then alternatively computes u_1^{n+1} and u_2^{n+1}, $n = 0, 1, 2, ...$, as follows:

$$\begin{cases} \Delta u_1^{n+1} = 0, & \text{in } \Omega_1, \\ u_1^{n+1} = g, & \text{on } \partial\Omega_1 \backslash \Gamma_1, \\ u_1^{n+1} = u_2^n, & \text{on } \Gamma_1, \end{cases} \qquad \begin{cases} \Delta u_2^{n+1} = 0, & \text{in } \Omega_2, \\ u_2^{n+1} = g, & \text{on } \partial\Omega_2 \backslash \Gamma_2, \\ u_2^{n+1} = u_1^{n+1}, & \text{on } \Gamma_2. \end{cases}$$

Schwarz presented the earliest alternating Schwarz method to prove Dirichlet's principle. Later, with the development of parallel computing and computational mathematics, a multitude of interesting DDM have been developed. Formally, we can extend the Schwarz method to a general differential operator with boundary condition as follows:

$$\begin{aligned} \mathcal{L}(u) &= f, \quad \text{in}\, \Omega, \\ \mathcal{B}(u) &= g, \quad \text{on}\, \partial\Omega, \end{aligned} \tag{2}$$

where $\mathcal{L}$ can be a negative Laplace operator, $-\Delta$, or any reasonable operator, $\mathcal{B}$ can be $\frac{\partial}{\partial \mathbf{n}} + I$ or any boundary condition, and f, g are two given functions. We always presume that the problem (2) is well-posed. Suppose that $\mathcal{D}$ is the operator of an artificial interface transmission condition, e.g., Dirichlet, Neumann or Robin. We then have a parallel general DDM with domain setting in Figure 1 listed in Algorithm 1. Different from the Schwarz alternating method, the presented Algorithm 1 can be implemented parallel.

Algorithm 1 A parallel Domain Decomposition Method for Two Subdomains in Figure 1

```
1: Give an initial guess $w_1^0, w_2^0$ along $\Gamma_1$
2: for i=1, ... do
3:     Solve for $u_1^i$
           $$\begin{aligned} \mathcal{L}(u_1^i) &= f, \quad \text{in } \Omega_1, \\ \mathcal{B}(u_1^i) &= g, \quad \text{on } \partial\Omega_1 \backslash \Gamma_1, \\ \mathcal{D}(u_1^i) &= w_1^{i-1}, \quad \text{on } \Gamma_1. \end{aligned} \tag{3}$$
4:     $w_2^i \leftarrow \mathcal{D}\left(u_1^i(x_{\Gamma_2})\right)$
5:     Solve for $u_2^i$
           $$\begin{aligned} \mathcal{L}(u_2^i) &= f, \quad \text{in } \Omega_2, \\ \mathcal{B}(u_2^i) &= g, \quad \text{on } \partial\Omega_2 \backslash \Gamma_2, \\ \mathcal{D}(u_2^i) &= w_2^{i-1}, \quad \text{on } \Gamma_2, \end{aligned} \tag{4}$$
```

```
6:        $w_1^i \leftarrow \mathcal{D}\left(u_2^i(x_{\Gamma_1})\right)$
7:        **if** $\|w_1^i - w_1^{i-1}\|/\|w_1^i\| < tol_\Gamma$ **and** $\|w_2^i - w_2^{i-1}\|/\|w_2^i\| < tol_\Gamma$ **then**
8:            STOP
9:        **end if**
10:       **if** $||u_1^i - u_1^{i-1}||/||u_1^i|| < tol_\Omega$ **and** $||u_2^i - u_2^{i-1}||/||u_2^i|| < tol_\Omega$ **then**
11:           STOP
12:       **end if**
13:   **end for**
```

Here, in Algorithm 1, the stop strategy is the relative error of the current solution with respect to the previous one on the artificial interface or inside the subdomain less than a given tolerance tol_Γ or tol_Ω, respectively.

3. Physics-informed neural networks

We now introduce the physics-informed neural networks for solving PDE Raissi et al. (2019). In this paper, we consider deep fully connected feedforward neural networks. The entire neural network consists of $L+1$ layers, where layer 0 is the input layer and layer L is the output layer. Layers $0 < l < L$ are the hidden layers. All of the layers have an activation function, excluding the output layer. The activation function can take the form of Sigmoid, Tanh (hyperbolic tangents) or ReLU (rectified linear units).

We denote $d_0, d_1, ..., d_L$ as a list of integers, with d_0, d_L representing the lengths of input signal and output signal of the neural network. Define function $\mathbf{T}_l : \mathbb{R}^{d_l} \to \mathbb{R}^{d_{l+1}}$, $0 \le l < L$,

$$\mathbf{T}_l(\mathbf{x}) = \mathbf{W}_l\mathbf{x} + \mathbf{b}_l,$$

where σ is the activation function and $\theta := \{\mathbf{W}_l, \mathbf{b}_l\}$ represents the collection of all parameters. Thus, we can simply represent a deep fully connected feedforward neural network using the composite function $h(\cdot;\theta) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$,

$$h(\cdot;\theta) = \mathbf{T}_L \circ \sigma \circ \mathbf{T}_{L-1} \circ \cdots \circ \mathbf{T}_1 \circ \sigma \circ \mathbf{T}_0,$$

where σ is the activation function and $\theta := \{\mathbf{W}_l, \mathbf{b}_l\}$ represents the collection of all parameters.

Solving a general PDE such as (2) by DNN is a physics-informed minimization problem with the objective $\mathcal{M}(\theta)$ consisting of two terms as follows:

$$\theta^* = \text{argmin}_\theta\, \mathcal{M}(\theta) := \mathcal{M}_\Omega(\theta) + \mathcal{M}_{\partial\Omega}(\theta),$$

$$\mathcal{M}_\Omega(\theta) := \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\mathcal{L}\left(h(\mathbf{x}_f^i;\theta)\right) - f(\mathbf{x}_f^i)\right|^2,\ \ \mathcal{M}_{\partial\Omega}(\theta) := \frac{1}{N_g}\sum_{i=1}^{N_g}\left|\mathcal{B}\left(h(\mathbf{x}_g^i;\theta)\right) - g(\mathbf{x}_g^i)\right|^2, \tag{5}$$

where $\{\mathbf{x}_f^i\}_{i=1}^{N_f}$ and $\{\mathbf{x}_g^i\}_{i=1}^{N_g}$ are the collocation points in the inside and on the boundary, respectively. The domain term $\mathcal{M}_\Omega$ and boundary term $\mathcal{M}_{\partial\Omega}$ enforce the condition that the desired optimized neural network $h(\cdot;\theta^*)$ satisfies $\mathcal{L}(u) = f$ and $\mathcal{B}(u) = g$, respectively.

Gradient descent methods can be used to solve this kind of optimization problem; however, from the empirical point of view, more effective and efficient stochastic gradient descents with minibatches are recommended LeCun et al. (2012). With regard to the analysis of convergence of stochastic gradient descent, there are many early contributions, such as Bottou (1998); Kiwiel (2001).

We should note that the PINN actually provide another kind of discretization scheme for solving PDE. As in the cases of FEM or FDM, a basic and interesting question is as follows: is it possible to use $h(\mathbf{x};\theta)$ to approximate the solution of PDE? A well-known answer is as follows: if the solution $u(\mathbf{x})$ is bounded and continuous, then $h(\mathbf{x};\theta)$ can approximate $u(\mathbf{x})$ to any desired accuracy, given the increasing hidden neurons Hornik et al. (1989); Hornik (1991) and Cybenko (1989).

4. Deep domain decomposition methods

Now, it is natural to mix DDM with DNN. Here, we provide a general formulation. Suppose that we divide the problem (2) into S subproblems defined in S subdomains as follows:

$$\begin{aligned}
\mathcal{L}(u_s) &= f_s, \quad \text{in } \Omega_s, \\
\mathcal{B}(u_s) &= g_s, \quad \text{on } \partial\Omega_s \backslash \Gamma_s, \\
\mathcal{D}(u_s) &= \mathcal{D}(u_r), \quad \text{on } \Gamma_s
\end{aligned} \tag{6}$$

where f_s and g_s are the expressions of f and g in domain Ω_s, respectively, and Γ_s is the artificial interface of domain Ω_s to other subdomains. To simplify the discussion, we simply denote u_r as the solution of the neighboring subproblems of u_s. We also denote $\mathcal{D}$ as the operator of the artificial interface transmission condition.

We denote $h_s(\mathbf{x};\theta)$, $1 \leq s \leq S$ as the DNN we used for each subproblem (6), i.e., a surrogate of the solution u_s, and denote θ_s as network parameters of the k-th subproblem. That means different neural networks are used for different subproblems. In contrast with the minimization problem (5), we can define subminimization problems as follows:

$$\mathcal{M}_s(\theta;\mathbf{X}_s) := \mathcal{M}_{\Omega_s}(\theta;\mathbf{X}_{f_s}) + \mathcal{M}_{\partial\Omega_s\backslash\Gamma_s}(\theta;\mathbf{X}_{g_s}) + \mathcal{M}_{\Gamma_s}(\theta;\mathbf{X}_{\Gamma_s}), \tag{7}$$

where

$$\begin{aligned}
\mathcal{M}_{\Omega_s}(\theta;\mathbf{X}_{fs}) &:= \frac{1}{N_{f_s}} \sum_{i=1}^{N_{f_s}} \left| \mathcal{L}\left(h_s(\mathbf{x}^i_{f_s};\theta)\right) - f(\mathbf{x}^i_{f_s}) \right|^2, \\
\mathcal{M}_{\partial\Omega_s\backslash\Gamma_s}(\theta;\mathbf{X}_{g_s}) &:= \frac{1}{N_{g_s}} \sum_{i=1}^{N_{g_s}} \left| \mathcal{B}\left(h_s(\mathbf{x}^i_{g_s};\theta)\right) - g(\mathbf{x}^i_{g_s}) \right|^2, \\
\mathcal{M}_{\Gamma_s}(\theta;\mathbf{X}_{\Gamma_s}) &:= \frac{1}{N_{\Gamma_s}} \sum_{i=1}^{N_{\Gamma_s}} \left| \mathcal{D}\left(h_s(\mathbf{x}^i_{\Gamma_s};\theta)\right) - \mathcal{D}\left(h_r(\mathbf{x}^i_{\Gamma_s};\theta)\right) \right|^2,
\end{aligned} \tag{8}$$

with $\mathbf{X}_{f_s} := \{\mathbf{x}^i_{f_s}\}_{i=1}^{N_{f_s}}$, $\mathbf{X}_{g_s} := \{\mathbf{x}^i_{g_s}\}_{i=1}^{N_{g_s}}$ and $\mathbf{X}_{\Gamma_s} := \{\mathbf{x}^i_{\Gamma_s}\}_{i=1}^{N_{\Gamma_s}}$ representing the collocation points in the inside, on the local boundary and on the interface for the s-th subproblem, respectively. Denote $\mathbf{X}_s := \{\mathbf{X}_{f_s}, \mathbf{X}_{g_s}, \mathbf{X}_{\Gamma_s}\}$ as the set of whole data. To rearrange the training data $\mathbf{X}_s$ to several minibatches at each epoch, we only consider the data inside the subdomain. That means, at each epoch, we first randomly rearrange $\mathbf{X}_{f_s}$ into m_s disjoint parts $\{\mathbf{X}^1_{f_s}, \mathbf{X}^2_{f_s}, ..., \mathbf{X}^{m_s}_{f_s}\}$, and then let $\mathbf{X}^k_s := \left\{\mathbf{X}^k_{f_s}, \mathbf{X}_{g_s}, \mathbf{X}_{\Gamma_s}\right\}$ be the k-th minibatch[1]. For simplicity, we denote $\mathbf{W}_s := \mathcal{D}\left(h_r(\mathbf{X}_{\Gamma_s};\theta)\right)$

1. We numerically found that this strategy can make the learning process stable. Here we give a heuristic explanation. Since the number of N_{g_s} or N_{Γ_s} is much smaller than that of N_{f_s}, once we directly rearrange the total training data,

as the information that would be transported to the objective subproblem labelled by s from the neighboring subproblems labelled by r. The DeepDDM algorithm for subproblem (6) is given in algorithm 2.

Algorithm 2 DeepDDM for the s-th subproblem (6), $1 \leq s \leq S$.

```
 1: Construct $\mathbf{X}_s$;
 2: Initial parameters $\theta_s^0$ and interface information $\mathbf{W}_s^0$ along $\Gamma_s$;
 3: for $i = 1, 2, ...$ do
 4:     Set $\theta_s^{i,0} := \theta_s^{i-1}$;                                    ▷ Start DDM iteration
 5:     for $j = 1, 2, ...$ do
 6:         Set $\theta_s^{i,j} := \theta_s^{i,j-1}$;                              ▷ Start DL iteration
 7:         Rearrange randomly training data $\{\mathbf{X}_s^k\}_{k=1}^{m_s}$;
 8:         for $k = 1, 2, ..., m_s$ do
 9:             Update $\theta_s^{i,j}$: $\theta_s^{i,j} \leftarrow \theta_s^{i,j} - \epsilon \nabla_{\theta_s^{i,j}} \mathcal{M}_s(\theta_s^{i,j}; \mathbf{X}_s^k)$;     ▷ Update on minibatch
10:         end for
11:         if $\left|\mathcal{M}_s(\theta_s^{i,j}; \mathbf{X}_s) - \mathcal{M}_s(\theta_s^{i,j-\eta}; \mathbf{X}_s)\right| \Big/ \left|\mathcal{M}_s(\theta_s^{i,j}; \mathbf{X}_s)\right| < tol_{\mathcal{M}}$ then
12:             BREAK;
13:         end if
14:     end for
15:     Set $\theta_s^i := \theta_s^{i,j}$;
16:     Interchange the interface information $\mathbf{W}_s^i \leftarrow \mathcal{D}\left(h_r(\mathbf{X}_{\Gamma_s}; \theta_s^i)\right)$;
17:     if $\left\|\mathbf{W}_s^i - \mathbf{W}_s^{i-1}\right\| \Big/ \left\|\mathbf{W}_s^i\right\| < tol_\Gamma, \ \forall\, s$ then
18:         STOP;
19:     end if
20:     if $\left\|h_s\left(\mathbf{X}_{f_s}; \theta_s^i\right) - h_s\left(\mathbf{X}_{f_s}; \theta_s^{i-1}\right)\right\| \Big/ \left\|h_s\left(\mathbf{X}_{f_s}; \theta_s^i\right)\right\| < tol_\Omega, \ \forall\, s$ then
21:         STOP;
22:     end if
23: end for
```

Some remarks are shown as follows.

- The loop in Step 3 implements the iteration among subproblems, i.e., domain decomposition; the loop in Step 5 implements the iteration of the objective function in each subproblem, i.e., deep learning; and the loop in Step 8 implements the iteration on each minibatch;
- The stop criterion for subproblem optimization, shown in Step 11, is the relative error of current loss and historical loss at step $j-\eta$. Here once a subproblem reach the stop criterion, it will break the loop 5 and wait for other subproblems to interchange the interface information;
- The stop criterion for DDM, shown in Step 17 and Step 20, is the relative error of solution on interface collocation points and inside collocation points, respectively. Here the DeepDDM algorithm will stop after all the subproblems reach stop criteria.

we may have no boundary information in some minibatches. However, for solving a general PDE, the boundary information is quite essential. A more detailed theoretical or numerical analysis would strengthen this point. Considering it's somehow independent with the key inspirit of DeepDDM, we ignore the related numerical discussion.

Computational cost: Obviously, benefiting from DDM, DeepDDM can be implemented in parallel. Suppose we fixed the total number of training data, denoted by N. Once we divide the domain to S subdomains, then the training data for each subproblem is about N/S. For a fixed network architecture, the training time of one epoch is $O(N/S)$. Denote the number of training epoch (DL iteration in algorithm 2) for each subproblem as J_s. Denote the number of DDM iteration as I_S which usually depends on the number of subdomains. Then given a fixed network architecture, the total computational cost of DeepDDM is $O(\max_s J_s \cdot I_S \cdot N/S)$. For considering the dependency of computation cost on network architecture, it highly depends on the computational device (GPU) and the detailed operators in the network. For example, if the model size located at a reasonable range and we use NVIDIA V100, for a fixed number of training data, the learning time for one epoch varies little.

5. Numerical examples

In this section, we present a set of systematic numerical results of elliptic problems, including a model problem and an interface problem with a curved interface. The results of the model problem in different subdomain cases are compared with respect to various network architectures and overlapping size conditions in Section 5.1. In Section 5.2, the results of the interface problem with discontinuous coefficients and different coefficient ratio increases are shown, further demonstrating the effectiveness and robustness of our approach. Benefiting from the mesh-free strength, the implementation of domain decomposition can be easily handled even in cases of curved interfaces or more complex geometries. DeepDDM not only attains promising accuracy but also preserves some properties of DDM existing in conventional discrete strategy.

Setting: In our cases, network architectures are simply set as having equal units for each layer. The $Layers$ used later indicates the hidden layers in the network, and $Units$ indicates the units per layer. We chose Tanh as activation function and stochastic gradient descent using Adam Kingma and Ba (2014) to update the optimization method. The mini-batches are set as 64 in this paper, if not otherwise specified. For the choice of learning rate, the initial learning rates and decay rates change for different situations. More specifically, learning rates initially range from 10^{-3} to 10^{-2} and decay every $1-100$ steps with a base of $0.99-0.999$. The training data herein are all randomly selected, and the test data used to calculate errors are regularly sampled by row and column, with 40,000 in all. The number of training data indicates the amount of that in the whole domain, and so, the amount of data used in a subdomain is equal to the number divided by the number of subdomains in the model problem or by the subdomain area ratio in the interface problem.

5.1. Model problem

The Poisson equation is a simplified form of an important equation derived from engineering and physical problems. The effectiveness of applying algorithms to the Poisson equation is a necessary prerequisite for algorithms applied to more complicated equations.

We consider a Poisson equation with Dirichlet boundary conditions:

$$\begin{aligned} -\nabla \cdot (\nabla u) &= f, \quad \text{in } \Omega := [0, \pi] \times [0, 1], \\ u(x, y) &= g, \quad \text{on } \partial\Omega. \end{aligned} \tag{9}$$

Table 1: Relative $\mathbb{L}^2$ error between the predicted and the exact solution for different numbers of hidden layers and different numbers of units per layer. Here, the total numbers of training data are fixed to $N_g = 50 \times 4$ and $N_f = 2500$.

$Layers$ \ $Units$	10	20	30	40	50	100
2	4.1e-3	2.6e-3	1.4e-3	1.5e-3	1.0e-3	1.0e-3
3	1.7e-3	1.2e-3	5.8e-4	3.2e-4	4.3e-4	3.7e-4
4	1.7e-4	1.0e-3	7.1e-4	5.0e-4	5.0e-4	2.1e-4

Table 2: Relative $\mathbb{L}^2$ error between the predicted and the exact solution for different numbers of training data. The network architecture is fixed to $Layers = 3$, $Units = 50$.

N_g \ N_f	100	400	900	1600	2500	10000
10×4	7.3e-3	2.8e-3	3.4e-3	6.3e-3	9.3e-3	2.5e-3
30×4	4.5e-3	2.0e-3	1.9e-3	1.4e-3	1.8e-3	1.4e-3
100×4	3.9e-3	9.8e-4	7.1e-4	6.2e-4	4.7e-4	1.2e-4

To obtain an analytic solution, we take, for example, $u_* = sin(2x)e^y$, and substitute it into (9) to compute f and g. We denote the numerical solution by u_h and define the relative $\mathbb{L}^2$ error:

$$\mathcal{E} := \left(\frac{\sum_{i=1}^{n} |u_* - u_h|^2}{\sum_{i=1}^{n} |u_*|^2} \right)^{\frac{1}{2}}.$$

To begin with, we present the results of solving (9) by physics-informed neural networks. The stop criterion of the inner iteration training process is $\left|(\mathcal{M}^n - \mathcal{M}^{n-100})/\mathcal{M}^n\right| < 10^{-3}$, or that we exceeded the maximum number of allowed iterations, set as 50,000 here, when the strategy without domain decomposition is used. Although there is no theory to ensure that the optimization method certainly converges on the global minimum point, our practical experiences indicate that good results would be obtained if the problem is well-posed, the network is large enough, and the amount of data is large.

In Table 1, relative $\mathbb{L}^2$ errors between the predicted and the exact solution are presented for different network architectures, while the total numbers of training points are maintained as $N_g = 50 \times 4$ and $N_f = 2500$, respectively. As expected, if the number of $Layers$ is fixed, more $Units$ corresponds with smaller relative $\mathbb{L}^2$ errors. Moreover, for a fixed number of $Units$, relative errors would also be reduced when the number of $Layers$ increases. The general trend suggests that the numerical solutions obtained will be more accurate upon increasing expression capacity of the network. This observation is in agreement with the results of Raissi et al. (2019); Yang and Perdikaris (2019).

In Table 2, we fixed the network architecture, $Layers = 3$ and $Units = 50$ and recorded the change of relative $\mathbb{L}^2$ error under different numbers of training data. As expected, as the amount of training data is increased, a more accurate numerical solution is obtained. Furthermore, the output is more sensitive to N_g than N_f. This observation is in agreement with the results of Raissi et al. (2019); Yang and Perdikaris (2019).

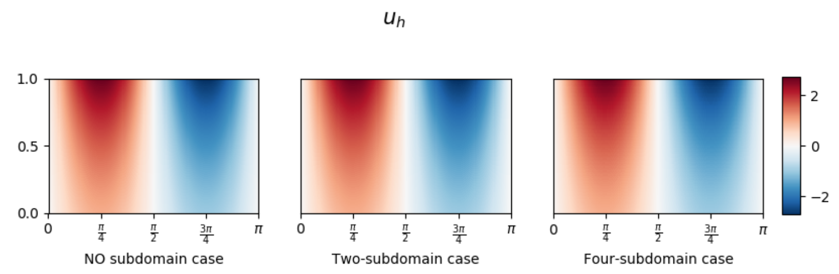


Figure 2: The numerical solutions in the whole domain for different cases. Here, the network architecture is fixed to $Layers = 3$, $Units = 50$; the total numbers of training data are fixed to $N_g = 50 \times 4$, $N_\Gamma = 50$ and $N_f = 2500$; and $\delta = 0.1$ for domain decomposition cases.

Furthermore, to illustrate the effectiveness of DeepDDM and investigate its properties, we present the results of the two-subdomain case and multi-subdomain case. In practice, we set $tol_\Omega = tol_\Gamma = 10^{-2}$. The stop criterion of the inner iteration training process is that $\left|(\mathcal{M}^n - \mathcal{M}^{n-100})/\mathcal{M}^n\right| < 5 * 10^{-3}$, or that the maximum number of allowed iterations, set as 10,000, is exceeded when DeepDDM is used herein. Let δ be overlapping. For the two-subdomain case, we set $\Omega_1 := [0, \frac{\pi}{2} + \frac{\delta}{2}] \times [0,1]$, $\Omega_2 := [\frac{\pi}{2} - \frac{\delta}{2}, \pi] \times [0,1]$; however, the numerical solutions are simply defined as follows:

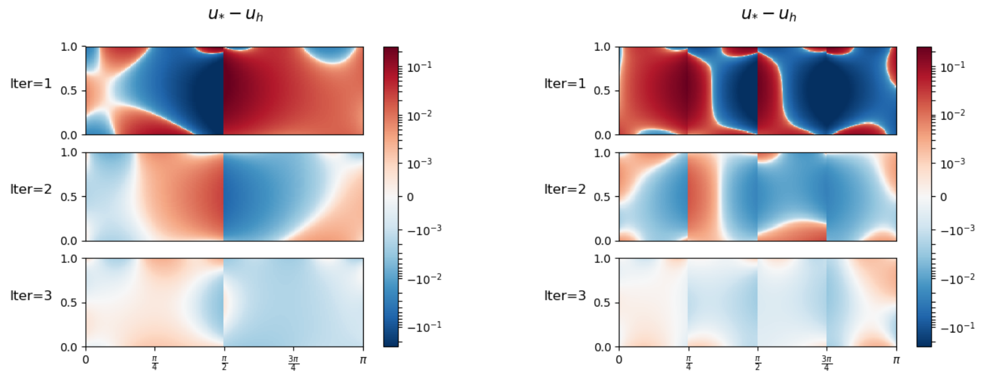
$$u_h = \begin{cases} u_{h,1}, \ x \in (0, \frac{\pi}{2}), \\ u_{h,2}, \ x \in (\frac{\pi}{2}, \pi). \end{cases}$$

For four-subdomain case, we set $\Omega_1 := [0, \frac{\pi}{4} + \frac{\delta}{2}] \times [0,1]$, $\Omega_2 := [\frac{\pi}{4} - \frac{\delta}{2}, \frac{\pi}{2} + \frac{\delta}{2}] \times [0,1]$, $\Omega_3 := [\frac{\pi}{2} - \frac{\delta}{2}, \frac{3\pi}{4} + \frac{\delta}{2}] \times [0,1]$, $\Omega_4 := [\frac{3\pi}{4} - \frac{\delta}{2}, \pi] \times [0,1]$, however the numerical solutions are simply defined as follows,

$$u_h = \begin{cases} u_{h,1}, \ x \in (0, \frac{\pi}{4}), \\ u_{h,2}, \ x \in (\frac{\pi}{4}, \frac{\pi}{2}), \\ u_{h,3}, \ x \in (\frac{\pi}{2}, \frac{3\pi}{4}), \\ u_{h,4}, \ x \in (\frac{3\pi}{4}, \pi). \end{cases}$$

First, for the same situation, same network $Layers = 3$, $Units = 50$ and same training data $N_g = 50 \times 4$, $N_\Gamma = 50$ and $N_f = 2500$, the predicted solutions of the Poisson equation for three cases are presented in Figure 2. The results obtained from these cases are close to each other. In Figure 3, the network is fixed to $Layers = 3$, $Units = 20$, the numbers of data to $N_g = 50 \times 4$, $N_\Gamma = 50$ and $N_f = 2500$, and overlapping $\delta = 0.8$; the errors of solutions in the whole domain are presented for different outer iterations for the two-subdomain case and four-subdomain case. It is clearly visible that the errors of solutions are relatively larger after the first outer iteration. However, the numerical solution quickly approximates the exact solution as outer iterations increase. After the third outer iteration, the algorithm reaches the stop criterion, and the errors of solutions reach the order of 10^{-3}. The performance of DeepDDM in the multi-subdomain case is still promising, and the entire convergence process can be clearly observed.

In Table 3, the relative $\mathbb{L}^2$ errors and the number of outer iterations are shown for different network architectures for the two-subdomain case and four-subdomain case. The accuracy acquired by DeepDDM in table 3 is almost the same as that of the results in Table 1 in various scales of network



(a) Two-subdomain case (b) Four-subdomain case

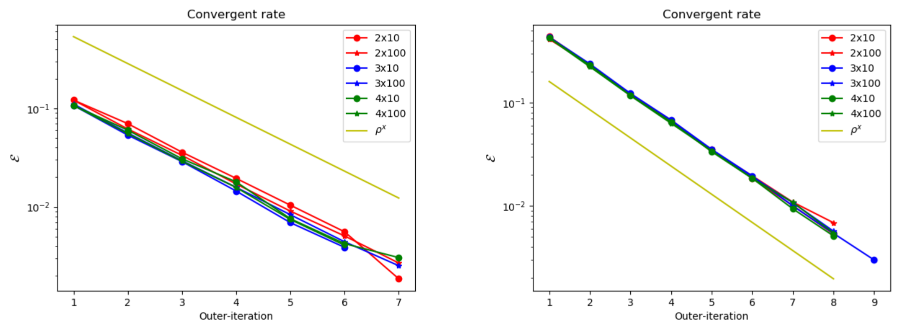
Figure 3: The error $u_* - u_h$ in the whole domain for outer iterations. Here, for both cases, the network architecture is fixed to $Layers = 3, Units = 20$; the total numbers of training data are fixed to $N_g = 50 \times 4$, $N_\Gamma = 50$ and $N_f = 2500$; and $\delta = 0.8$.

Table 3: Relative $\mathbb{L}^2$ error and the number of outer iterations for different network architectures under two-subdomain case and four-subdomain case. The figures in parentheses are the numbers of outer iterations. For the two-subdomain case, the total numbers of training are fixed to $N_g = 100\times4$, $N_\Gamma = 100$ and $N_f = 10000$. For the four-subdomain case, the total numbers of training are fixed to $N_g = 50 \times 4$, $N_\Gamma = 50$ and $N_f = 2500$. Here, $\delta = 0.2$ for both cases.

No. of subdomains	$Units$ / $Layers$	10	20	30	40	50	100
Two	2	1.9e-3(7)	4.9e-3(6)	2.7e-3(7)	2.6e-3(7)	4.0e-3(6)	2.7e-3(7)
	3	3.9e-3(6)	2.9e-3(7)	1.0e-3(8)	2.4e-3(7)	2.6e-3(7)	2.5e-3(7)
	4	3.1e-3(7)	2.6e-3(7)	2.5e-3(7)	2.5e-3(7)	4.3e-3(6)	4.1e-3(6)
Four	2	5.4e-3(8)	6.1e-3(8)	5.4e-3(8)	5.6e-3(8)	5.9e-3(8)	6.9e-3(8)
	3	3.0e-3(9)	5.5e-3(8)	5.8e-3(8)	5.9e-3(8)	4.9e-3(8)	5.7e-3(8)
	4	5.1e-3(8)	6.1e-3(8)	5.7e-3(8)	6.0e-3(8)	5.7e-3(8)	5.5e-3(8)

architectures. Otherwise, if we focus on the number of outer iterations, the general trend indicates that the number of outer iterations remain approximately constant no matter how the network architecture changes. This is consistent with our expectation that the number of outer iterations is independent of the function space, i.e., the network architecture. Further, all numbers of outer iterations obtained in the four-subdomain case are larger than those of the two-subdomain case. This observation coincides with the conventional DDM. As the number of subdomains increases, the number of outer iterations required will increase a little. The table indicates that DeepDDM not only has the same accuracy as PINN, but also can be implemented in parallel. Different training data are employed for different cases, but the impact on our results is extremely limited.

Further results can be obtained as long as we plot the process of error convergence. Firstly, if Fourier analysis is used for the domain decomposition model of (9), we would have the analytic



(a) Two-subdomain case (b) Four-subdomain case

Figure 4: The change in relative $\mathbb{L}^2$ error along with out-iteration in different network architectures. Here, the training data are fixed to $N_g = 100 \times 4$, $N_\Gamma = 100$ and $N_f = 10000$ for the two-subdomain case, the training data are fixed to $N_g = 50 \times 4$, $N_\Gamma = 50$ and $N_f = 2500$ for the four-subdomain case, and $\delta = 0.2$ for both.

domain decomposition convergence factor:

$$\rho := e^{-k_{min}\delta}, \tag{10}$$

where k_{min} is the minimum Fourier frequency, and δ is the overlap size Gander (2006). In our case, $k_{min} = \pi$. The process of error convergence and the analytic convergence factor are shown in Figure 4. Only a few cases are selected; however, other network architecture cases exhibit similar results. It is clearly observed that the convergence rates of numerical results are always close to that of analysis, regardless of what kind of network architecture is used, in two-subdomain and four-subdomain cases. This observation coincides with the proposition that the convergence factor depends only on overlap size and domain size.

A more detailed assessment of the effects of overlap sizes is presented in Table 4. In particular, we present relative $\mathbb{L}^2$ errors and outer iterations, for both the two-subdomain case and four-subdomain case, with different overlaps from 0.05 to 0.8. If we fix the overlap and compare results from the two-subdomain case and the four-subdomain case, the latter requires more iterations than the former. With the increase in the overlapping domain, the outer iterations decline monotonically whether for the two-subdomain case or four-subdomain case. Moreover, the accuracy would decline as the overlapping domain diminishes, which occurs on account of the decrease of training data in the overlapping domain and the randomness of samples. It is noteworthy that these observations coincide with the conventional DDM. Additionally, Figure 5 presents the comparison of convergence rates for different sizes of overlap under the two-subdomain case and the four-subdomain case. The line with different colors represents the case with different overlap sizes. The dashed line and the dotted line respectively represent the two-subdomain case and the four-subdomain case, and the analytic convergence factors are plotted with solid lines. The figure clearly displays that the numerical convergence rate is closely related to overlap size rather than the number of subdomains. From the analytical point of view, as long as the overlapping domain expands, the convergence factor would be smaller, and so, the algorithm converges faster in numerical experiments.

The experiments described above demonstrate that DeepDDM effectively inherits the nature of DDM. Similarly to conventional DDM, as the number of subdomains increases, the necessary

Table 4: Relative $\mathbb{L}^2$ error and the number of outer iterations for different sizes of overlap and decomposition. Here, the total numbers of training are fixed to $N_g = 50$, $N_\Gamma = 50$ and $N_f = 2500$ and the network architecture is fixed to $Layers = 3, Units = 20$.

Overlap	2x1 decompositions	4x1 decompositions
0.05	1.6e-2(13)	2.1e-2(22)
0.1	9.0e-3(9)	1.7e-2(13)
0.2	2.6e-3(7)	5.5e-3(8)
0.4	7.6e-4(5)	3.3e-3(5)
0.8	9.5e-4(3)	6.9e-4(3)

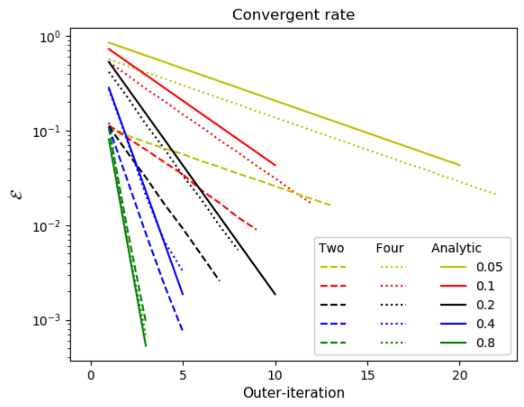


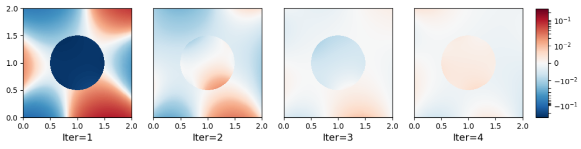
Figure 5: The convergence rates under different overlap sizes and in different subdomain cases. Here, the network architecture is fixed to $Layers = 3, Units = 20$, and the total numbers of training are fixed to $N_g = 50 \times 4$, $N_\Gamma = 50$ and $N_f = 2500$. Dashed line: the two-subdomain case. Dotted line: the four-subdomain case. Solid line: analytic convergence factors.

iterations also escalate, and the convergence factor shrinks along with the expansion of overlapping domains. We can perhaps introduce coarse space correction, just like the conventional manner, to improve the performance of DeepDDM. Moreover, we find that the method of sampling affects the computing results to some extent, and so, the adaptive strategy should be considered and adopted. However, we leave these questions for future work.

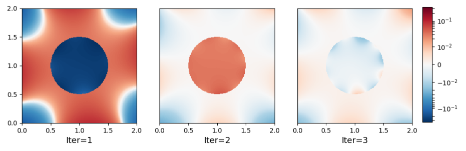
5.2. Interface problem with discontinuous coefficients

The example aims to highlight the ability of our approach in handling anisotropic coefficients and a curved interface condition in the governing PDE. To this end, we consider the case of a steady-state diffusion problem:

$$\begin{aligned} -\nabla\cdot(a(\mathbf{x})\nabla u) &= f, && \text{in } \Omega := [0,2]\times[0,2],\\ \mathcal{B}(u) &= g, && \text{on } \partial\Omega,\\ [u] = \left[a(\mathbf{x})\frac{\partial u}{\partial \mathbf{n}}\right] &= 0, && \text{on } \Gamma := \{(x,y)|(x-1)^2+(y-1)^2 = 0.25\}, \end{aligned} \tag{11}$$



(*a*) $\alpha = 2$.



(*b*) $\alpha = 20$.

Figure 6: The error $u_* - u_h$ in the whole domain for outer iterations with (a) $\alpha = 2$ and (b) $\alpha = 20$ in the interface problem. Here, the net architecture is fixed to $Layers = 3$ and $Units = 20$, and training data are fixed to $N_g = 50 \times 4$, $N_\Gamma = 50 \times 4$ and $N_f = 2500$.

where $\mathbf{n}$ is the outer normal of the interface and the scalar coefficient $a(\mathbf{x})$ is the piecewise constant function

$$a(\mathbf{x}) = \begin{cases} 1, \text{ for } \mathbf{x} \in \Omega_1 := \{(x,y)|(x-1)^2 + (y-1)^2 < 0.25\}, \\ \alpha, \text{ for } \mathbf{x} \in \Omega_2 := \Omega \backslash \Omega_1. \end{cases}$$

The α symbolize the ratio of coefficient. As α is enlarged, the heterogeneity strengthens, which means that it is more challenging to approximate the exact solution numerically. In the case of strong heterogeneity in the material, corresponding to a large coefficient ratio, domain decomposition is a recommended approach. We choose the exact solution as follows,

$$u = \begin{cases} \alpha((x-1)^2 + (y-1)^2) - 0.25(\alpha - 1), & \text{in } \Omega_1, \\ (x-1)^2 + (y-1)^2, & \text{in } \Omega_2. \end{cases}$$

so that the force term f and boundary condition g in (11) can be easily obtained. It is also easy to verify that the above functions exactly satisfy the interface condition in (11). Here, we choose the boundary operator $\mathcal{B}$ as the identity operator, which means that the Dirichlet boundary condition is employed. Using physical coupling conditions between the subdomains, the equation can be written (in a weak sense) in a multidomain formulation,

$$\begin{cases} -\Delta u_1 = f, & \text{in } \Omega_1, \\ u_1 = u_2, & \text{on } \Gamma, \end{cases} \qquad \begin{cases} -\alpha \Delta u_2 = f, & \text{in } \Omega_2, \\ u = g, & \text{on } \partial\Omega_2 \backslash \Gamma, \\ \alpha \dfrac{\partial u_2}{\partial \mathbf{n}} = \dfrac{\partial u_1}{\partial \mathbf{n}}, & \text{on } \Gamma. \end{cases}$$

The above equations in two subdomains imply that we set $\mathcal{D}$ the identity operator (Dirichlet boundary condition) in Ω_1 and the outer normal operator (Neumann boundary condition) in Ω_2 during applying Alogrithm 2.

To assess the strength of our approach, and by focusing on the curved interface and anisotropic coefficient problems, we present a set of numerical results for various conditions. The results given demonstrate the effectiveness and the rapid convergence of DeepDDM in various situations. Moreover, some properties of conventional DDM are also retained in DeepDDM for interface problems, e.g. the iteration is independent of function space; the number of iterations is positively correlated with the ratio of coefficients. In practice, the tolerance of the stop criterion for DeepDDM is set as $tol_\Gamma = tol_\Omega = 10^{-2}$. Additionally, the stop criterion of the training process in each subproblem is that $\left|(\mathcal{M}^n - \mathcal{M}^{n-100})/\mathcal{M}^n\right| < 5 \times 10^{-3}$, or that the maximum number of allowed iterations, 10,000, is reached.

In the following example, our setup aims to highlight the robustness and effectiveness of the proposed algorithm, even in cases of small network architecture. The changes in the error of numerical solutions and exact solutions in the whole domain for $\alpha = 2$ and $\alpha = 20$ are given in Figure 6(a)subfigure and Figure 6(b)subfigure, respectively. Our training dataset consists of $N_g = 50 \times 4$, $N_\Gamma = 50 \times 4$ and $N_f = 2500$. All of the training data are randomly sampled.

We chose to represent the latent function u_h using a 3-layer network with 20 units per layer. As shown by Figure 6, the numerical solution quickly approaches the exact solution, along with the subdomain information interchanged on the interface. Obtained from the test data in the $\mathbb{L}^2$ norm, the relative errors under $\alpha = 2$ and $\alpha = 20$ are, respectively, $2.6 \cdot 10^{-3}$ and $2.4 \cdot 10^{-3}$. It is noted that the elementary Dirichlet-Neumann interface condition is used in the strongly discontinuous coefficient problems, implying overlap equal to 0.

The relative $\mathbb{L}^2$ errors and outer-iterations for $\alpha = 2$ and $\alpha = 20$ are summarized in Table 5. Here, the total number of training data is kept fixed at $N_g = 50 \times 4$, $N_\Gamma = 50 \times 4$ and $N_f = 2500$, respectively. The key observation here is that as the numbers of $Layers$ and $Units$ increase; i.e., the capacity of the network to approximate more complex functions is enlarged, the numerical accuracy is improved. When it comes to iterations, they are independent of the network architecture, which is similar to the observation from the previous example. It is remarkable that the errors of the solution for specific network architecture with $\alpha = 20$ are almost equal to the errors of corresponding situations in the $\alpha = 2$ case. Additionally, we also observe that the iteration needed in the $\alpha = 20$ case is relatively smaller than that in the $\alpha = 2$ case. In other words, when α becomes larger within a proper range, the number of outer iterations needed is attenuated. A similar observation has been found in Gander and Dubois (2015).

Respecting the convergence rate of DeepDDM, Figure 7 illustrates the change in relative $\mathbb{L}^2$ errors along with outer iterations in various network architectures at $\alpha = 2$ and $\alpha = 20$. We selected some network architecture cases in Table 5 and then plotted the error as the iterations increase. According to the yellow line, we can conjecture that the numerical convergence factor is about 0.5. The theoretical analysis of this interface problem is still an open problem.

6. Conclusions

We have introduced DeepDDM, a novel framework bridging deep learning and domain decomposition, to approximate solutions of PDE according to given equation information. The presented approach showcases a series of promising results for a model problem and an interface problem. These numerical results demonstrate that the convergence rate of DeepDDM applied to the Poisson equation is close to the analytical value of DDM. DeepDDM maintains some properties of conventional DDM: for instance, the outer-iteration depends on the size overlap and the number of subdomains instead of function space. Furthermore, DeepDDM can easily handle PDE with curve interface and heterogeneity.

This work is seminal in combining deep learning and domain decomposition, and it presents some experiments to provide insights for theoretical study. The work does not pertain to a stack of fundamental topics behind the approach presented. Take the following questions as examples. What is the convergence rate of deep learning solving PDE? What is the standard to design the best network architecture? How do we sample the training data for the best result? Some of these questions are open problems in this field; however, it is still significant to explore the properties of

Table 5: Relative $\mathbb{L}^2$ errors and the number of outer iterations for different network architectures. These figures in parentheses indicate the number of outer iterations. Here, the total number of training data is fixed to $N_g = 50 \times 4$, $N_\Gamma = 50 \times 4$ and $N_f = 2500$.

Cases	Units / Layers	10	20	30	40	50	100
$\alpha = 2$	2	4.4e-3(4)	3.3e-3(4)	1.8e-3(5)	6.1e-4(4)	2.6e-3(4)	4.4e-3(3)
	3	6.5e-3(4)	2.6e-3(4)	7.7e-3(3)	1.7e-3(5)	4.4e-3(4)	2.0e-3(5)
	4	2.6e-3(5)	1.9e-3(4)	2.1e-3(4)	1.4e-3(5)	4.0e-3(3)	3.5e-3(3)
$\alpha = 20$	2	1.2e-2(4)	1.5e-2(2)	5.9e-3(3)	5.5e-3(3)	3.3e-3(3)	2.2e-3(3)
	3	1.4e-3(3)	2.4e-3(3)	5.1e-3(3)	3.8e-3(3)	5.0e-3(2)	5.3e-3(3)
	4	7.1e-3(3)	3.9e-3(3)	5.4e-3(3)	5.3e-3(2)	1.4e-3(2)	2.7e-3(2)

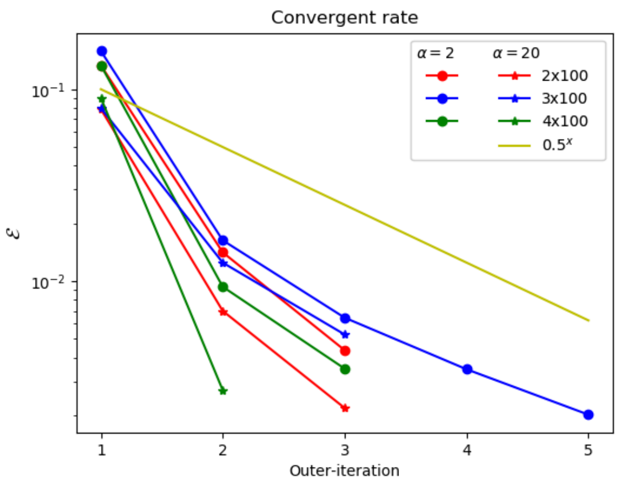


Figure 7: The change in relative $\mathbb{L}^2$ error along with out-iteration in different network architectures and different α. Here, the training data are kept fixed to $N_g = 50 \times 4$, $N_\Gamma = 50 \times 4$ and $N_f = 2500$.

DeepDDM. Future works should consider applying DeepDDM with the coarse space correction and higher-order interface conditions.

Acknowledgments

This research was funded by National Natural Science Foundation of China with grant 11831016, Beijing Nova Program of Science and Technology under Grant Z191100001119129 and National Natural Science Foundation of China with grant 11671074.

References

Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL http://leon.bottou.org/papers/bottou-98x. revised, oct 2012.

Tony F Chan and Tarek P Mathew. Domain decomposition algorithms. *Acta numerica*, 3:61–143, 1994.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Victorita Dolean, Pierre Jolivet, and Frédéric Nataf. *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*, volume 144. SIAM, 2015.

Maksymilian Dryja and Olof B Widlund. Some domain decomposition algorithms for elliptic problems. In *Iterative methods for large linear systems*, pages 273–291. Elsevier, 1990.

Vikas Dwivedi, Nishant Parashar, and Balaji Srinivasan. Distributed physics informed neural network for data-efficient solution to partial differential equations. *arXiv preprint arXiv:1907.08967*, 2019.

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

Charbel Farhat, Michel Lesoinne, Patrick LeTallec, Kendall Pierson, and Daniel Rixen. Feti-dp: a dual–primal unified feti method—part i: A faster alternative to the two-level feti method. *International journal for numerical methods in engineering*, 50(7):1523–1544, 2001.

Martin J Gander. Optimized schwarz methods. *SIAM Journal on Numerical Analysis*, 44(2):699–731, 2006.

Martin J Gander and Olivier Dubois. Optimized schwarz methods for a diffusion problem with discontinuous coefficient. *Numerical Algorithms*, 69(1):109–144, 2015.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2):251–257, 1991.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

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

Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

Krzysztof C Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001.

Axel Klawonn, Martin Lanser, and Oliver Rheinbach. Nonlinear feti-dp and bddc methods. *SIAM Journal on Scientific Computing*, 36(2):A737–A765, 2014.

Martin Heinrich Lanser. *Nonlinear feti-dp and bddc methods*. PhD thesis, Universität zu Köln, 2015.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Ke Li, Kejun Tang, Tianfan Wu, and Qifeng Liao. D3m: A deep domain decomposition method for partial differential equations. *IEEE Access*, 2019.

Pierre-Louis Lions. On the schwarz alternating method. i. In *First international symposium on domain decomposition methods for partial differential equations*, volume 1, page 42. Paris, France, 1988.

Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

Nam Mai-Duy and Thanh Tran-Cong. Mesh-free radial basis function network methods with domain decomposition for approximation of functions and numerical solution of poisson's equations. *Engineering Analysis with Boundary Elements*, 26(2):133–156, 2002.

Tarek Mathew. *Domain decomposition methods for the numerical solution of partial differential equations*, volume 61. Springer Science & Business Media, 2008.

Clemens Pechstein and Clark R Dohrmann. A unified framework for adaptive bddc. *Electron. Trans. Numer. Anal*, 46:273–336, 2017.

M Raissi, P Perdikaris, and GE 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.

Keith Rudd, Gianluca Di Muro, and Silvia Ferrari. A constrained backpropagation approach for the adaptive solution of partial differential equations. *IEEE transactions on neural networks and learning systems*, 25(3):571–584, 2013.

Hermann Amandus Schwarz. *Ueber einen Grenzübergang durch alternirendes Verfahren*. Zürcher u. Furrer, 1870.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

Barry Smith, Petter Bjorstad, and William Gropp. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge university press, 2004.

Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.