Domain Decomposition Algorithms for Physics-Informed Neural Networks

Hyea Hyun Kim[1] and Hee Jun Yang[2]

1 Introduction

Domain decomposition algorithms are widely used as fast solutions of algebraic equations arising from discretization of partial differential equations. The original algebraic equations are partitioned and solved in each subdomain combined with an iterative procedure. The resulting solution for the original algebraic equations is then obtained from the iterative procedure. In such approaches, the convergence often gets slow as more subdomains are introduced. To accelerate the convergence, a global coarse problem is formed and combined in the iterative procedure. We refer [9] for a general introduction to domain decomposition algorithms.

Recently, there have been developed many successful approaches to solve partial differential equations using deep neural networks, see [1, 8, 7, 5]. The advantage of these new approaches is that they can be used for partial differential equations without much concern on discretization methods suitable for the given problem. On the other hand, a suitable design of the neural network and a suitable choice of data sets for training the parameters are important for these new approaches. In general, the network can be large and the data set can be also large. The parameter training then becomes inefficient and even may encounter numerical instability.

The purpose of this study is to develop domain decomposition algorithms for solutions of partial differential equations using deep neural networks. The idea is similar to the classical domain decomposition methods. The problem is solved using independent smaller neural networks iteratively and the smaller neural networks are trained as solutions of local problems, that are restriction of the original problem to smaller subdomains. In previous pioneering studies by [3, 4], the same idea is used but there has been no study for accelerating the convergence of the iterative scheme. In this work, an additional global coarse network is introduced and it is trained as

[1]Department of Applied Mathematics and Institute of Natural Sciences, Kyung Hee University, Korea. `hhkim@khu.ac.kr` ·[2]Department of Mathematics, Kyung Hee University, Korea. `yhjj109@khu.ac.kr`

a solution of the global problem using a coarse data set. The global coarse network is then used to accelerate the convergence of the iterative solution obtained from the independent smaller neural networks. The smaller neural networks and global coarse network are trained in each iteration. Their parameter training can be done in parallel. Among several neural network approaches, we will consider the PINN (Physics Informed Neural Network) method by [7]. Our domain decomposition approach can be applied to other methods by [1, 8, 5] as well.

In this work we report the first successful result for parallel algorithms for PINN using both local networks and one global coarse network. The introduction of the global coarse network is noble and it accelerate the convergence of the iteration. Numerical results also present that the use of the global coarse network makes the parallel algorithm scalable, i.e., the number of iterations is robust to the increase of the number of subdomains.

This paper is organized as follows. In Section 2, we introduce the method by PINN for solving partial differential equations and in Section 3 we propose a two-level overlapping domain decomposition algorithm for solving partial differential equations utilizing the PINN approach. In Section 4, numerical results are presented for a model elliptic problem in two dimensions and conclusions are given.

2 Physics informed neural networks (PINN)

We will introduce the physics-informed neural networks (PINN) which are trained to solve supervised learning tasks in order to satisfy any given laws of physics described by partial differential equations, see [7]. We consider a general differential operator with a boundary condition,

$$\begin{aligned} \mathcal{L}(u) &= f, \quad \text{in } \Omega, \\ \mathcal{B}(u) &= g, \quad \text{on } \partial\Omega, \end{aligned} \tag{1}$$

where $\mathcal{L}$ can be a differential operator defined for a function u and $\mathcal{B}$ describes a given boundary condition on u, and f, g are given functions. We assume that the model problem in (1) is well-posed and the solution u exists. We then approximate the solution u in (1) by a neural network, $U(x;\theta)$, that can be trained by minimizing the cost function $\mathcal{J}(\theta)$ consisting of the two terms

$$\mathcal{J}(\theta) = \mathcal{J}_{X_\Omega}(\theta) + \mathcal{J}_{X_{\partial\Omega}}(\theta),$$

where

$$\begin{aligned} \mathcal{J}_{X_\Omega}(\theta) &:= \frac{1}{|X_\Omega|} \sum_{x \in X_\Omega} |\mathcal{L}(U(x;\theta)) - f(x)|^2, \\ \mathcal{J}_{X_{\partial\Omega}}(\theta) &:= \frac{1}{|X_{\partial\Omega}|} \sum_{x \in X_{\partial\Omega}} |\mathcal{B}(U(x;\theta)) - g(x)|^2. \end{aligned}$$

In the above, X_D denotes the collection of points chosen from the region D and $|X_D|$ denotes the number of points in the set X_D. The cost function $\mathcal{J}_{X_\Omega}(\theta)$ and $\mathcal{J}_{X_{\partial\Omega}}(\theta)$ are designed so that the optimized neural network $U(x;\theta)$ satisfies the equations in (1) derived from physics laws.

3 A two-level overlapping algorithm for PINN

We consider the following model elliptic problem in two dimensional domain Ω,

$$\begin{aligned} -\triangle u &= f \text{ in } \Omega, \\ u &= g \text{ on } \Omega. \end{aligned} \tag{2}$$

We propose an iterative scheme to find its solution u by using overlapping subdomain partition, $\{\Omega_i\}_i$, with an overlapping width δ. For a given $u^{(n)}$, we solve the following problem in each subdomain Ω_i to find $u_i^{(n+1)}$,

$$\begin{aligned} -\triangle u_i^{(n+1)} &= f \text{ in } \Omega_i, \\ u_i^{(n+1)} &= u^{(n)} \text{ on } \partial\Omega_i, \\ u_i^{(n+1)} &= u^{(n)} \text{ in } \overline{\Omega} \setminus \Omega_i. \end{aligned} \tag{3}$$

Using $u_i^{(n+1)}$, the next iterate is given by

$$u^{(n+1)} = (1 - N\tau)u^{(n)} + \tau \sum_{i=1}^{N} u_i^{(n+1)}, \tag{4}$$

where N denotes the number of subdomains and τ denotes the relaxation parameter. Let N_c be the maximum number of subdomains sharing the same geometric position in Ω. With $\tau \leq 1/N_c$, $u^{(n)}$ converges to the solution u of (2) under a suitably chosen space of functions, see [10, 9, 2, 6]. We can rewrite the above iteration formula as follows: for any x in Ω

$$u^{(n+1)}(x) = (1 - |s(x)|\tau)u^{(n)}(x) + \tau \sum_{i \in s(x)} u_i^{(n+1)}(x), \tag{5}$$

where $s(x)$ denotes the set of subdomain indices sharing x and $|s(x)|$ denotes the number of elements in the set $s(x)$. We introduce

$$\widehat{u}^{(n+1)}(x) := \frac{1}{|s(x)|} \sum_{i \in s(x)} u_i^{(n+1)}(x)$$

and rewrite the above iteration formula into

$$u^{(n+1)}(x) = (1 - |s(x)|\tau)u^{(n)}(x) + |s(x)|\tau\widehat{u}^{(n+1)}(x).$$

Using this formula, we can see that $\widehat{u}^{(n+1)}(x)$ also converges to $u(x)$.

For i in $s(x)$, the solution $u_i^{(n+1)}(x)$ is updated after solving the local problem in (3). We thus define $U_i(x;\theta_i^{(n+1)})$ as a neural network function to approximate $u_i^{(n+1)}(x)$ in each Ω_i. Using the method of PINN, we can find the optimal parameters $\theta_i^{(n+1)}$. Using them, we define

$$\widehat{U}^{(n+1)}(x) := \frac{1}{|s(x)|}\sum_{i\in s(x)} U_i(x;\theta_i^{(n+1)}). \tag{6}$$

We now propose the following one-level method:

Algorithm 1: One-level method (input: $U^{(0)}$, output: $\widehat{U}^{(n+1)}$)

Step 0: Let $U^{(0)}(x)$ be given and $n = 0$.

Step 1: Find $\theta_i^{(n+1)}$ in $U_i(x;\theta_i^{(n+1)})$ for

$$\begin{aligned} -\triangle u &= f \text{ in } \Omega_i, \\ u &= U^{(n)} \text{ on } \partial\Omega_i. \end{aligned}$$

Step 2: Update $U^{(n+1)}$ at each data set $X_{\partial\Omega_i}$ as, see (6),

$$U^{(n+1)}(x) = (1 - \tau|s(x)|)U^{(n)}(x) + \tau|s(x)|\widehat{U}^{(n+1)}.$$

Step 3: Go to **Step 1** with $n = n + 1$ or set the output as $\widehat{U}^{(n+1)}$ if the stopping condition is met.

Using sufficiently large enough neural network functions $U_i(x;\theta_i^{(n)})$, we can approximate $u_i^{(n+1)}(x)$ and $\widehat{U}^{(n+1)}(x)$ will thus approximate $\widehat{u}^{(n+1)}(x)$. Since $\widehat{u}^{(n+1)}(x)$ converges to $u(x)$, $\widehat{U}^{(n+1)}(x)$ will converge to $u(x)$. We note that if one wishes to take $U^{(n+1)}(x)$ as the final output then one needs to store all the parameters $\theta_i^{(m)}$ for all previous steps m. In addition, the evaluation of $U^{(n+1)}(x)$ at any given point x can be very expensive. We thus take $\widehat{U}^{(n+1)}(x)$ as the final solution in our algorithm. We will only need to store the parameters $\theta_i^{(n+1)}$ at the final step. Since the local problems in the above algorithm are solved by the PINN method, the function $\widehat{U}^{(n+1)}(x)$ needs to be evaluated at x in the data set $X_{\partial\Omega_i}$. In our algorithm, we only store these function values at each iteration and use them when we solve the local problems (3) using the PINN method.

As we can see in numerical results provided in Section 4, the convergence of the one-level algorithm gets slower as more subdomains are introduced in the partition. We thus improve the one-level algorithm by enriching the boundary condition $U^{(n)}$ with a suitable coarse correction term. For a given $U^{(n)}$, we consider the following global problem:

$$
\begin{aligned}
-\triangle u_c^{(n)} &= f \text{ in } \Omega_\delta,\\
-\triangle u_c^{(n)} &= -\triangle U^{(n)} \text{ in } \Omega \setminus \Omega_\delta,\\
u_c^{(n)} &= g \text{ on } \partial\Omega,
\end{aligned} \tag{7}
$$

where Ω_δ denotes the overlapping region of the subdomain partition $\{\Omega_i\}_i$. For the solution $u_c^{(n)}$, we can obtain the following error equation,

$$
\begin{aligned}
-\triangle(u - u_c^{(n)}) &= 0 \text{ in } \Omega_\delta,\\
-\triangle(u - u_c^{(n)}) &= -\triangle(u - U^{(n)}) \text{ in } \Omega \setminus \Omega_\delta,\\
u - u_c^{(n)} &= 0 \text{ on } \partial\Omega.
\end{aligned} \tag{8}
$$

From the above error equation, we have $u - u_c^{(n)}$ with smaller errors than $u - U^{(n)}$. We will then find a coarse correction term $U_c^{(n)}(x;\theta_c^{(n)})$ that approximates $u_c^{(n)}$ with the parameters $\theta_c^{(n)}$ determined by the PINN method. Using the coarse correction term, for $\alpha > 0$ we set

$$\widetilde{U}^{(n)} = (1-\alpha)U^{(n)} + \alpha U_c^{(n)}$$

and use it when we evaluate the boundary condition for the local problems in (3). We note that when we find $\theta_c^{(n)}$ using the PINN method we will only need to evaluate $-\triangle U^{(n)}$ at the data set $X_{\Omega\setminus\Omega_\delta}$ without the need to store the parameters $\theta_i^{(m)}$ for all previous steps m.

We now summarize the two-level method:

Algorithm 2: Two-level method (input: $U^{(0)}$, output: $\widehat{U}^{(n+1)}$)

Step 0: Let $U^{(0)}(x)$ be given and $n = 0$.

Step 1-1: Find $U_c^{(n)}(x;\theta_c^{(n)})$ for (7) and set

$$\widetilde{U}^{(n)}(x) = (1-\alpha)U^{(n)}(x) + \alpha U_c^{(n)}(x;\theta_c^{(n)}).$$

Step 1-2: Find $\theta_i^{(n+1)}$ in $U_i(x;\theta_i^{(n+1)})$ for

$$
\begin{aligned}
-\triangle u &= f \text{ in } \Omega_i,\\
u &= \widetilde{U}^{(n)} \text{ on } \partial\Omega_i.
\end{aligned}
$$

Step 2: Update $U^{(n+1)}(x)$ at each data set $X_{\partial\Omega_i}$ as, see (6),

$$U^{(n+1)}(x) = (1 - \tau|s(x)|)U^{(n)}(x) + \tau|s(x)|\widehat{U}^{(n+1)}.$$

Step 3: Go to **Step 1-1** with $n = n + 1$ or set the output as $\widehat{U}^{(n+1)}$ if the stopping condition is met.

4 Numerical results

We perform numerical results of the proposed two algorithms for the model problem in (2) with f and g given according to the known exact solution $u(x, y)$ and with Ω as a unit rectangular domain. The domain Ω is partitioned into uniform rectangular subdomains with an overlapping width δ. For the iterates $U^{(n)}$, we stop the iteration when the relative l^2-error between the two successive iterates is less than 5×10^{-3}. When training parameters $\theta_i^{(n)}$ and $\theta_c^{(n)}$, we stop the iteration when the relative errors for cost function values between 100 steps is less than 10^{-4} or when the number of iterations is more than the maximum number of epochs, that is set as 5000. For local problems, we use neural network functions as a two block Resnet with each block consisting of 10 hidden layers and with Tanh as the activation function, that give 921 parameters $\theta_i^{(n)}$ for each local problem. To train the parameters, we use 200 data points for X_{Ω_i} and 40 data points for $X_{\partial\Omega_i}$. For the coarse problem, we use the same network and the same size of data sets.

In our method, we have two parameters τ and α. For τ, we can set τ as less than or equal to $1/N_c$ and α as a number between 0 and 1. When $\alpha = 0$, the two-level algorithm is identical to the one-level algorithm. With $\alpha > 0$, the method is enhanced with the coarse correction term.

In Table 1, we report the performance of the proposed method with various α and N for the exact solution $u(x, y) = \sin(\pi x) \sin(\pi y)$. The relative L^2-errors to the exact solution and the number of iterations are presented. We set τ as $1/4$, note $N_c = 4$. Without the coarse correction term, i.e., $\alpha = 0$, the one-level method shows that the number of iterations increases as increasing N. For the other choices of $\alpha (> 0)$, the coarse correction term accelerates the convergence and the number of iterations seems robust to the increase of the number of subdomains.

Table 1: The performance with $\tau = 1/4$ depending on α and N (the subdomain partition): the numbers are relative L^2-errors to the exact solution and the numbers inside the parenthesis are the number of iterations.

N	$\alpha = 0$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 1$
2×2	0.0098(22)	0.0073(14)	0.0117(19)	0.0082(21)
3×3	0.0282(34)	0.0243(17)	0.0260(14)	0.0310(14)
4×4	0.0402(52)	0.0070(18)	0.0237(10)	0.0431(8)
5×5	0.0769(67)	0.0255(21)	0.0298(13)	0.0392(10)

We compare our Algorithm 1 and that in the previous study by [4]. Under the same setting with Table 3 of [4], we apply our Algorithm 1 and obtain much less iteration and more accurate solutions, see Table 2.

To show the advantage of partitioning the problem, we consider a more difficult problem with the exact solution given by

$$u(x, y) = 100x(1 - x)y(1 - y) \sin((x - 0.5)(y - 0.5)/0.05). \tag{9}$$

Table 2: The performance of Algorithm 1 under the same setting for the model in [4, Table 3]: relative L^2-errors and the number of iterations (numbers inside the parenthesis) depending on the number of layers (L), and the number of units (U).

N	L \ U	10	20	30	40	50	100
4	2	0.0040(2)	0.0037(2)	0.0030(2)	0.0043(2)	0.0023(2)	0.0029(2)
4	3	0.0042(2)	0.0034(2)	0.0046(2)	0.0030(2)	0.0037(2)	0.0061(2)
4	4	0.0075(2)	0.0046(2)	0.0038(2)	0.0045(2)	0.0060(2)	0.0047(2)

To approximate the highly oscillatory solution with high contrast, we use a single neural network with its number of parameters as 9109 and with 2000 interior points and 400 boundary points for training the parameters using 250000 epochs. With this, we solve the model problem in the whole domain $\Omega = (0\,1)^2$. For the same model problem, we partition the domain into 9 overlapping subdomains and employ a smaller neural network with 921 number of parameters. For the global coarse network, we use the same number of parameters. For training parameters in both local and coarse neural networks, 200 interior points and 40 boundary points are used. The computation time and the accuracy of trained solutions are compared in Table 3. We can observe the advantage of partitioning with much less computation time and less errors than the single domain case. When the local solutions are solved in parallel, the computation time can be further reduced. For the analysis of computational time, we let T_s be the training time for one local or coarse neural network, and T be the training time for the single neural network of the whole domain. Let $iter$ be the number of iterations in our Algorithm 2. Assuming that the local networks are trained in parallel, the total computation time becomes $iter \times 2T_s$. With a proper size of local and coarse neural networks, the computation time T_s can become much smaller than T and the total computation time is thus expected to be much smaller than T.

Table 3: The performance of the proposed method for the model problem in (9): single domain and 9 subdomains with different α values, the numbers inside the parenthesis are the number of iterations.

	single domain	$\alpha = 0$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 1.0$
L^2-error	0.0754	0.0820 (93)	0.0623 (56)	0.0705 (61)	0.0793 (61)
time(sec)	289840	74702	49980	54442	54442

In conclusions, a two-level algorithm suitable for deep neural network architecture is proposed and tested. By partitioning the large deep neural network, the computation time is greatly reduced with a more accurate solution in our test example. More rigorous numerical study and convergence analysis will be done in a more complete paper.

Acknowledgments: The authors are supported by NRF-2019R1A2C1010090.

References

1. Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.*, 5(4):349–380, 2017.
2. Martin J. Gander. Schwarz methods over the course of time. *Electron. Trans. Numer. Anal.*, 31:228–255, 2008.
3. Ke Li, Kejun Tang, Tianfan Wu, and Qifeng Liao. D3m: A deep domain decomposition method for partial differential equations. *IEEE Access*, 8:5283–5294, 2019.
4. Wuyang Li, Xueshuang Xiang, and Yingxiang Xu. Deep domain decomposition method: Elliptic problems. In *Mathematical and Scientific Machine Learning*, pages 269–286. PMLR, 2020.
5. Zichao Long, Yiping Lu, and Bin Dong. PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network. *J. Comput. Phys.*, 399:108925, 17, 2019.
6. Jongho Park. Additive Schwarz methods for convex optimization as gradient methods. *SIAM J. Numer. Anal.*, 58(3):1495–1530, 2020.
7. M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.
8. Justin Sirignano and Konstantinos Spiliopoulos. DGM: a deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.
9. Andrea Toselli and Olof Widlund. *Domain decomposition methods—algorithms and theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2005.
10. Jinchao Xu and Ludmil Zikatanov. The method of alternating projections and the method of subspace corrections in Hilbert space. *J. Amer. Math. Soc.*, 15(3):573–597, 2002.