Extended physics-informed neural networks (XPINNs) : A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations

Ameya D. Jagtap* [1]
George Em Karniadakis [1]
[1]Division of Applied Mathematics, Brown University,
182 George Street,
Providence, RI 02912, USA.
ameyadjagtap@gmail.com, ameya_jagtap@brown.edu

Abstract

We propose a generalized space-time domain decomposition framework for the physics-informed neural networks (PINNs) to solve nonlinear partial differential equations (PDEs) on arbitrary complex-geometry domains. The proposed framework, named eXtended PINNs (*XPINNs*), further pushes the boundaries of both PINNs as well as conservative PINNs (cPINNs), which is a recently proposed domain decomposition approach in the PINN framework tailored to conservation laws. Compared to PINN, the XPINN method has large representation and parallelization capacity due to the inherent property of deployment of multiple neural networks in the smaller subdomains. Unlike cPINN, XPINN can be extended to any type of PDEs. Moreover, the domain can be decomposed in any arbitrary way (in space and time), which is not possible in cPINN. Thus, XPINN offers both space and time parallelization, thereby reducing the training cost more effectively. In each subdomain, a separate neural network is employed with optimally selected hyperparameters, e.g., depth/width of the network, number and location of residual points, activation function, optimization method, etc. A deep network can be employed in a subdomain with complex solution, whereas a shallow neural network can be used in a subdomain with relatively simple and smooth solutions. We demonstrate the versatility of XPINN by solving both forward and inverse PDE problems, ranging from one-dimensional to three-dimensional problems, from time-dependent to time-independent problems, and from continuous to discontinuous problems, which clearly shows that the XPINN method is promising in many practical problems. The proposed XPINN method is the generalization of PINN and cPINN approaches, both in terms of applicability as well as domain decomposition approach, which efficiently lends itself to parallelized computation. The XPINN code will be available on *https://github.com/AmeyaJagtap/XPINNs*.

Introduction

Recently deep neural networks (DNNs) have gained a lot of attention in the field of scientific machine learning (SciML). Thanks to their universal approximation properties, they can be exploited to construct alternative approaches for solving PDEs. In particular, they offer nonlinear approximation through the composition of hidden layers, which does not limit the approximation to the linear spaces. The training of a DNN based model (black-box surrogate model) usually requires a large amount of labeled data, which are often unavailable in many scientific applications. However, when the governing PDEs are known, their solutions can be learned in a physics-informed fashion with relatively small amounts of data. The physics-informed loss functions are constructed based on PDE residuals and the DNN is trained by minimizing this loss function, which, in turn, satisfies the governing physical laws. Recently, Raissi et al.(4) used automatic differentiation and proposed physics-informed neural networks (PINNs), where the PDE residual is incorporated into the loss function of fully-connected neural networks as a regularizer, thereby constraining the space of admissible solutions. In this setting, the problem of inferring solutions of PDEs is transformed into an optimization problem of the loss function. A major advantage of PINNs is providing a mesh-free algorithm as the differential operators in the governing PDEs are approximated by *automatic differentiation* (1). PINNs require a modest amount of data, which can be properly enforced in the loss function. PINNs can solve forward problems, where the solution of governing physical laws is inferred, as well as inverse problems, where unknown coefficients or even differential operators in the governing equations are identified. The PINNs has been applied extensively to solve various PDEs, see (10; 5) for more details.

One of the main limitations of PINNs is the large training cost, which can adversely affect their performance, especially for solving real-life applications, which require a PINN model to run in real-time. Therefore, it is crucial to accelerate the convergence of such models without sacrificing the performance. This issue was first addressed in the conservative PINN (cPINN) method for conservation laws (11) by employing the domain decomposition approach in PINN framework. Domain decomposition has been a fundamental development in standard numerical methods, e.g. finite elements, for solving the governing physical laws in the form of PDEs on parallel computers. In particular, the com-

putational domain is partitioned into several subdomains and these subdomains interact only through their shared boundaries where some appropriate continuity conditions are imposed. The global solution is recovered by a succession of solutions of independent sub-problems associated with the entire domain. The cPINN method can be extended to other types of equations (and not necessarily conservation laws) by employing properties of the solution/governing equation at hand. In this regard, we propose a generalized domain decomposition approach namely, the eXtended PINNs (*XPINNs*) (7). Like PINNs, XPINN can be used to solve any PDE. Also, it has all the advantages of cPINN like deployment of separate neural networks in each subdomain, efficient hyper-parameter adjustment for all networks, easy parallelization capacity, large representation capacity (due to multiple networks), etc. Moreover, it has the following additional merits over cPINN.

- **Generalized space-time domain decomposition** : The XPINN formulation offers highly irregular, convex/non-convex space-time domain decomposition with C^0 or more regular boundaries. Such domain decomposition can be useful in many applications like multi-physics/multi-scale computations, simulation involving non-smooth features such as cracks, shock waves, etc. Due to such decomposition, the XPINN method easily lends itself for space-time parallelization, thereby reducing training cost more effectively.
- **Extension to any differential equation(s)** : Unlike cPINN method, the XPINN based domain decomposition approach can be extended to any type of PDE(s), irrespective of its physical nature. This way, any differential equation can be solved efficiently and this makes XPINN method a genuinely generalized domain decomposition based PINN method, which can be easily parallelized.
- **Simple interface conditions** : Due to irregular domain decomposition, the interfaces form highly irregular shapes, especially in higher dimensions. In the XPINN, the interface conditions are very simple for any arbitrarily shaped interfaces, which does not need normal direction hence, the proposed approach can be easily extended to any complex geometry, even in higher dimensions. Moreover, such a simple interface condition is very useful in dynamic interface problems where the interface is moving.

Accurately solving complex systems of equations, especially in higher dimensions has become one of the biggest challenges in scientific computing. The advantages of XPINN makes it a suitable candidate for such complex simulations in higher dimensions, which in general require a large training cost.

Problem Formulation

The general form of a parametrized PDE is given by

$$\begin{aligned} \mathcal{L}_{\mathbf{x}}(u;\lambda) &= f(\mathbf{x}), \quad \mathbf{x} \in \Omega \subset \mathbb{R}^d, \\ \mathcal{B}_k(u) &= g_k(\mathbf{x}), \quad \mathbf{x} \in \Gamma_k \subset \partial\Omega \end{aligned} \tag{1}$$

for $k = 1, 2, \cdots, n_b$, where $\mathcal{L}_{\mathbf{x}}(\cdot)$ is the differential operator, u is the solution, $\lambda = \{\lambda_1, \lambda_2, \cdots\}$ are the model parameters, $\mathcal{B}_k(\cdot)$ can be Dirichlet, Neumann, or mixed boundary conditions and $f(\mathbf{x})$ is the forcing term. Note that for transient problems we consider time t as one of the component of $\mathbf{x}$, and the initial conditions can be simply treated as a particular type of boundary condition on the given spatio-temporal domain. The above setup encapsulates a wide range of problems in engineering and science. For equation (1), we define the residual $\mathcal{F}(u)$ as $\mathcal{F}(u) \coloneqq \mathcal{L}_{\mathbf{x}}(u;\lambda) - f(\mathbf{x})$.

Methodology

Mathematical setup for fully connected neural networks

Let $\mathcal{N}^L : \mathbb{R}^{D_i} \to \mathbb{R}^{D_o}$ be a feed-forward neural network of L layers and N_k neurons in k^{th} layer ($N_0 = D_i$, and $N_L = D_o$). The weight matrix and bias vector in the k^{th} layer ($1 \leq k \leq L$) are denoted by $\mathbf{W}^k \in \mathbb{R}^{N_k \times N_{k-1}}$ and $\mathbf{b}^k \in \mathbb{R}^{N_k}$, respectively. The input vector is denoted by $\mathbf{z} \in \mathbb{R}^{D_i}$ and the output vector at k^{th} layer is denoted by $\mathcal{N}^k(\mathbf{z})$ and $\mathcal{N}^0(\mathbf{z}) = \mathbf{z}$. We denote the activation function by Φ which is applied layer-wise along with the scalable parameters na^k, where n is the scaling factor. Layer-wise introduction of the additional parameters a^k changes the slope of activation function in each hidden-layer, thereby increasing the training speed. Moreover, these activation slopes can also contribute to the loss function through the slope recovery term, see (8; 9) for more details. Such locally adaptive activation functions enhance the learning capacity of the network, especially during the early training period. In the recent study, Jagtap et al. (3) proposed the Kronecker Neural Networks, which is a general framework for the neural network with adaptive activation functions. In particular, they proposed rowdy activation functions, which allows faster network training than the locally adaptive activation functions. But in this paper we are employing the locally adaptive activation functions. Mathematically, one can prove this by comparing the gradient dynamics of the adaptive activation function method against that of the fixed activation method. The gradient dynamics of the adaptive activation modifies the standard dynamics (fixed activation) by multiplying a conditioning matrix by the gradient and by adding the approximate second-order term. In this paper, we used scaling factor $n = 5$ for all hidden-layers and initialize $na^k = 1, \ \forall k$, see (9) for details.

The $(L-1)$-hidden layer feed-forward neural network is defined by

$$\mathcal{N}^k(\mathbf{z}) = \mathbf{W}^k \Phi(a^{k-1}\mathcal{N}^{k-1}(\mathbf{z})) + \mathbf{b}^k \in \mathbb{R}^{N_k}, \quad 2 \leq k \leq L \tag{2}$$

and $\mathcal{N}^1(\mathbf{z}) = \mathbf{W}^1\mathbf{z} + \mathbf{b}^1$, where in the last layer, the activation function is identity. By letting $\tilde{\mathbf{\Theta}} = \{\mathbf{W}^k, \mathbf{b}^k, a^k\} \in \mathcal{V}$ as the collection of all weights, biases, and slopes, and taking $\mathcal{V}$ as the parameter space, we can write the output of the neural network as

$$u_{\tilde{\mathbf{\Theta}}}(\mathbf{z}) = \mathcal{N}^L(\mathbf{z}; \tilde{\mathbf{\Theta}}),$$

where $\mathcal{N}^L(\mathbf{z};\tilde{\mathbf{\Theta}})$ emphasizes the dependence of the neural network output $\mathcal{N}^L(\mathbf{z})$ on $\tilde{\mathbf{\Theta}}$.

Extended Physics-Informed Neural Networks

In this section we shall discuss the XPINN methodology, which is basically a PINN method on the decomposed domains. Here we describe the basic terminology used in the follow up presentation.

- **Subdomains**: The subdomains $\Omega_q,\ q = 1, 2, \cdots N_{sd}$ refer to the non-overlapping subset of the whole computational domain Ω such that $\Omega = \bigcup_{q=1}^{N_{sd}} \Omega_q$ and $\Omega_i \cap \Omega_j = \partial\Omega_{ij},\ i \neq j$. N_{sd} represents the total number of subdomains. In the non-overlapping domain decomposition, the subdomains intersect only on their interface $\partial\Omega_{ij}$.
- **Interface**: The interface is the common boundary between two or more subdomains, where the corresponding Sub-Nets communicate with each other.
- **Sub-Net**: The sub-net also called as sub-PINNs refers to the individual PINN with their own set of optimized hyperparameters employed in each subdomain.
- **Interface Conditions**: These conditions are used to stitch the decomposed subdomains together in order to obtain a solution of the governing PDEs over the complete domain. Based on the nature of the governing equations, one or more interface conditions can be applied along the common interface such as solution continuity, flux continuity, etc.

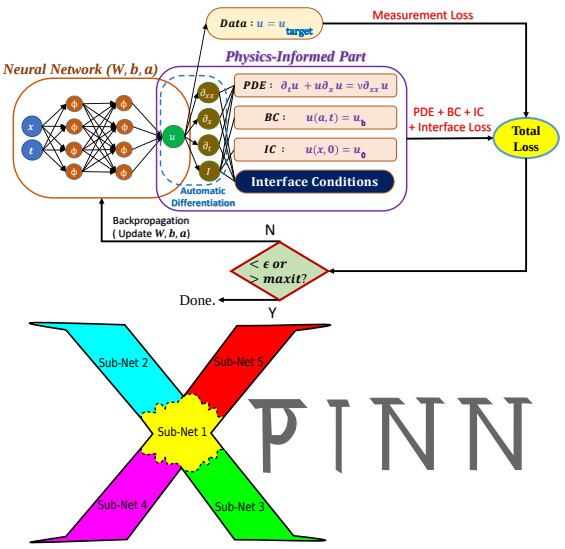

Figure 1: The top figure is the schematic of XPINN subnet employed in a subdomain where neural network and physics-informed part for viscous Burgers equation are shown. The bottom figure shows the irregularly shaped subdomain divisions in 'X'-shaped domain, where sub-net is employed in each subdomain and they are stitched together using the interface conditions. In this case, the domain boundary is shown by black continuous line, whereas the interface is shown by black dash line.

Figure 1 (top) shows a schematic of the XPINN Sub-Net, where along with DNN and PDE parts, additional interface conditions are also contributing to the loss function. The interface condition for XPINN includes the *residual continuity condition* in strong form as well as enforcing the average solution given by different Sub-Net's along the common interface. As discussed in the cPINN framework (11), for stability it is not necessary to enforce the average solution along the common interface, but the computational experiments reveal that it will drastically speed-up the convergence rate. Figure 1 (bottom) shows the schematic representation XPINNs where the 'X'-shaped domain is divided into irregular subdomains, and Sub-Net's are employed in each subdomain with different network architecture to obtain the solution of the same underlying PDE. Such domain decomposition also offers easy parallelization of the network, which is quite important in terms of achieving computational efficiency. XPINN has all the advantages of cPINN like parallelization capacity, large representation capacity, efficient choice for hyper-parameters like optimization method, activation function, depth or width of the network depending on some intuitive knowledge of the solution regularity in each subdomain, etc. In case of smooth zones, we can use a shallow network, whereas a deep neural network can be employed in a region where a complex solution is expected. Apart from this, there are various advantages of the XPINN approach over the cPINN method. Unlike cPINN, XPINN can be used to solve any type of PDEs and not necessarily conservation laws. In case of XPINN, there is no need to find the normal direction in order to apply normal flux continuity condition. This significantly reduces the complexity of the algorithm, especially in the case of large-scale problems with complex domains as well as for moving interface problems.

Consider the computational domain, which is divided into N_{sd} number of non-overlapping regular/irregular subdomains. In the XPINN framework, the output of the neural network in the q^{th} subdomain is given by

$$u_{\tilde{\mathbf{\Theta}}_q}(\mathbf{z}) = \mathcal{N}^L(\mathbf{z};\tilde{\mathbf{\Theta}}_q) \in \Omega_q, \quad q = 1, 2, \ldots, N_{sd}.$$

Then, the final solution is obtained as

$$u_{\tilde{\mathbf{\Theta}}}(\mathbf{z}) = \sum_{q=1}^{N_{sd}} u_{\tilde{\mathbf{\Theta}}_q}(\mathbf{z}) \cdot \mathbb{1}_{\Omega_q}(\mathbf{z}), \tag{3}$$

where the indicator function $\mathbb{1}_{\Omega_q}(\mathbf{z})$ is defined as

$$\mathbb{1}_{\Omega_q}(\mathbf{z}) := \begin{cases} 0 & \text{If } \mathbf{z} \notin \Omega_q, \\ 1 & \text{If } \mathbf{z} \in \Omega_q \backslash \text{Common interface in } \Omega_q, \\ \frac{1}{\mathcal{S}} & \text{If } \mathbf{z} \in \text{Common interface in } \Omega_q, \end{cases}$$

where $\mathcal{S}$ represent the number of subdomains intersecting along the common interface.

Subdomain loss function Let $\{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{u_q}}$, $\{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{F_q}}$ and $\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}}$ be the set of randomly selected training, residual, and the common interface points, respectively in the q^{th} subdomain. The N_{u_q}, N_{F_q}, and N_{I_q} represent the number of

training data points, the number of residual points, and the number of points on the common interface in the q^{th} subdomain, respectively.

Similar to PINN, the XPINN algorithm aims to learn a surrogate $u_q = u_{\tilde{\mathbf{\Theta}}_q}$, $q = 1, 2, \cdots, N_{sd}$ for predicting the solution $u = u_{\tilde{\mathbf{\Theta}}}$ of the given PDE over the entire computational domain using equation (3). The loss function of XPINN is defined subdomain-wise, which has a similar structure as the PINN loss function in each subdomain but is endowed with the interface conditions for stitching the subdomains together. For the forward problem, the loss function in the q^{th} subdomain is defined as

$$\begin{aligned}\mathcal{J}(\tilde{\mathbf{\Theta}}_q) = \; & W_{u_q}\, \text{MSE}_{u_q}(\tilde{\mathbf{\Theta}}_q; \{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{u_q}}) \\ & + W_{\mathcal{F}_q}\, \text{MSE}_{\mathcal{F}_q}(\tilde{\mathbf{\Theta}}_q; \{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{F_q}}) \\ & + W_{I_q}\, \underbrace{\text{MSE}_{u_{avg}}(\tilde{\mathbf{\Theta}}_q; \{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}})}_{\text{Interface condition}} \\ & + W_{I_{\mathcal{F}_q}}\, \underbrace{\text{MSE}_{\mathcal{R}}(\tilde{\mathbf{\Theta}}_q; \{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}})}_{\text{Interface condition}} \\ & + \underbrace{\text{Additional Interface Condition's}}_{\text{Optional}}, \end{aligned} \tag{4}$$

where $q = 1, 2, \cdots, N_{sd}$. The $W_{u_q}, W_{\mathcal{F}_q}, W_{I_{\mathcal{F}_q}}$ and W_{I_q} are the data mismatch, residual and interface (both, residual as well as average solution continuity along the interface) weights, respectively. The MSE is given for each term by

$$\text{MSE}_{u_q}(\tilde{\mathbf{\Theta}}_q; \{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{u_q}}) = \frac{1}{N_{u_q}} \sum_{i=1}^{N_{u_q}} \left| u^{(i)} - u_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^{(i)}) \right|^2,$$

$$\text{MSE}_{\mathcal{F}_q}(\tilde{\mathbf{\Theta}}_q; \{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{F_q}}) = \frac{1}{N_{F_q}} \sum_{i=1}^{N_{F_q}} \left| \mathcal{F}_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{F_q}^{(i)}) \right|^2,$$

$$\text{MSE}_{u_{avg}}(\tilde{\mathbf{\Theta}}_q; \{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}}) = \sum_{\forall q^+} \left(\frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| u_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{I_q}^{(i)}) - \left\{\left\{ u_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{I_q}^{(i)}) \right\}\right\} \right|^2 \right),$$

$$\text{MSE}_{\mathcal{R}}(\tilde{\mathbf{\Theta}}_q; \{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}}) = \sum_{\forall q^+} \left(\frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| \mathcal{F}_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{I_q}^{(i)}) - \mathcal{F}_{\tilde{\mathbf{\Theta}}_{q^+}}(\mathbf{x}_{I_q}^{(i)}) \right|^2 \right),$$

where the terms MSE_{u_q} and $\text{MSE}_{\mathcal{F}_q}$ are the data mismatch and residual losses, and $\mathcal{F}_{\tilde{\mathbf{\Theta}}_q} := \mathcal{F}(u_{\tilde{\mathbf{\Theta}}_q})$ represent the residual of the governing PDEs in the q^{th} subdomain. The $\text{MSE}_{\mathcal{R}}$ is the residual continuity condition on the common interface given by two different neural networks on subdomains q and q^+, respectively; the superscript $+$ over q represents the neighbouring subdomain(s). Both $\text{MSE}_{\mathcal{R}}$ and $\text{MSE}_{u_{avg}}$ terms are defined for all neighbouring subdomain(s) q^+, which is shown in their respective expressions by the summation sign over all q^+. The average value of u along the common interface is given by $\left\{\left\{ u_{\tilde{\mathbf{\Theta}}_q} \right\}\right\} = u_{avg} := \frac{u_{\tilde{\mathbf{\Theta}}_q} + u_{\tilde{\mathbf{\Theta}}_{q^+}}}{2}$ (assuming that along the common interface only two subdomains intersect). The interface conditions ensures the information from the one subdomain can be propagated throughout the neighboring subdomains. These conditions also play an important role in the convergence of subdomains where no training data is available.

Remark 3: In any domain decomposition method, interface conditions play an important role not only to stitch the subdomains together but also in terms of convergence. More specifically, the type of interface conditions decides the solution regularity across the interface, which can affect the convergence rate. In the proposed XPINN method, the enforcement of the average solution gives C^0 solution continuity across interface. Moreover, the residual continuity property can theoretically enforce the solution regularity in the classical sense, i.e., the solution across the interface is sufficiently continuous such that it satisfies its governing PDE, which is computed using AD. Therefore, XPINN can be used to solve any differential equations on decomposed domains. Apart from these conditions, *additional interface conditions* such as flux continuity, C^k solution continuity ($k > 0$) etc can be imposed depending on the type of the PDEs and the interface orientation. As an example, for conservation laws, both normal spatial flux and residual continuity conditions can be imposed on spatially divided subdomains, which makes the XPINN method a generalized domain decomposition method.

Remark 4: The weights $W_{u_q}, W_{\mathcal{F}_q}, W_{I_{\mathcal{F}_q}}$ and W_{I_q} play an important role in the convergence of the minimizer. These weights can be chosen dynamically, leading to faster convergence compared to static weights. However, such dynamic weights put additional computational burden, which can be significant in case of multiple loss functions based methods like cPINN and XPINN.

Remark 5: In case of a smooth solution, the enforcement of the average solution is similar to the solution continuity condition. However, in case of discontinuous solution, which we can expect in case of the hyperbolic conservation laws, such enforcement imposes the adjacent networks to satisfy the average value of the discontinuous solution along the interface.

Remark 6: A sufficient number of interface points (N_{I_q}) must be taken while stitching the subdomains together. This is important for faster convergence of the algorithm, especially for the internal subdomains, which does not have any training data points. We can also use the knowledge of solutions along the interface lines obtained from each subnetwork to properly select the locations of interface points.

Optimization Method We seek to find $\tilde{\mathbf{\Theta}}_q^*$ that minimizes the loss function $\mathcal{J}(\tilde{\mathbf{\Theta}}_q)$ in each subdomain. Even though there is no theoretical guarantee that the above mentioned procedure converges to a global minimum, our experiments indicate that as long as the given PDE is well-posed and has a unique solution, the XPINN formulation is capable of achieving an accurate solution provided that a sufficiently expressive network and a sufficient number of residual points are used. There are several optimization algo-

rithms available to minimize the loss function. The stochastic gradient descent (SGD) method is the widely used optimization method. In SGD, a small set of points are randomly sampled to find the direction of the gradient in every iteration. The SGD algorithm works well to avoid bad local minima during training of DNN under one point convexity property. In particular, we shall use the ADAM optimizer which is one version of SGD (6).

Remark 7: Note that due to highly non-convex nature of the XPINN loss function, it is very challenging to locate its global minimum. However, for several local minima, values of the loss function are comparable and the accuracy of the corresponding predicted solutions is similar. These are the instances of the so-called *operator mimicking phenomenon*, where the strong non-convexity of the loss function may lead to equally good multiple local minima.

Remark 8: In terms of coding, we can employ the same XPINN code to solve both forward as well as inverse problems, since for an inverse problem code we only need to add the additional model parameters involved in the governing PDEs to the list of the parameters to be optimized without modifying the other parts of forward problem code. The XPINN codes are written in Python, and the deep learning framework Tensorflow is used to take advantage of its inbuilt automatic differentiation capability.

Computational Results and Discussion

Viscous Burgers Equation

The one-dimensional viscous Burgers equation is given by

$$u_t + uu_x = \nu u_{xx}, \quad x \in \Omega \subset \mathbb{R}, \;\; t > 0$$

with initial condition $u(x, 0) = -\sin(\pi x)$, boundary conditions $u(-1, t) = u(1, t) = 0$ and $\nu = 0.01/\pi$. The analytical solution can be obtained using the Hopf-Cole transformation, see Basdevant et al., (2) for more details. The nonlinearity in the convection term develops a very steep solution due to the small value of diffusion coefficient ν.

The computational space-time domain is divided into two subdomains as shown in figure 2, where the internal subdomain boundary is in the shape of a dolphin. Table 1 shows the network architecture in both subdomains. Two different activation functions are used in the two subdomains. The learning rate is 0.0008. The values of data mismatch, residual and interface terms weights are $W_{u_q} = 20, W_{\mathcal{F}_q} = 1, W_{I_{\mathcal{F}_q}} = 1$ and $W_{I_q} = 20$, respectively, and the number of interface points on the interface is 310. The number of boundary and initial training data points is 300.

Subdomain number	1	2
# Layers	6	7
# Neurons	20	25
# Residual points	7000	3000
Adaptive Activation function	tanh	sin

Table 1: One-dimensional viscous Burgers equation: Neural network architecture in each subdomain.

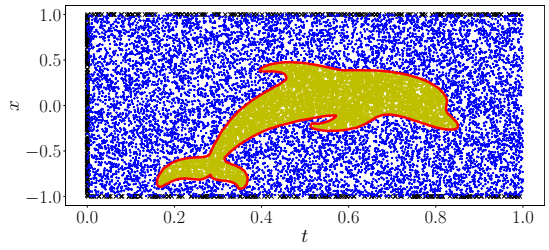

Figure 2: One-dimensional viscous Burgers equation: Residual points in subdomain 1 (blue stars) and subdomain 2 (yellow circles). Red line represent the dolphin shaped interface, which separates the two subdomains, and black cross represents the training data points from initial and boundary conditions.

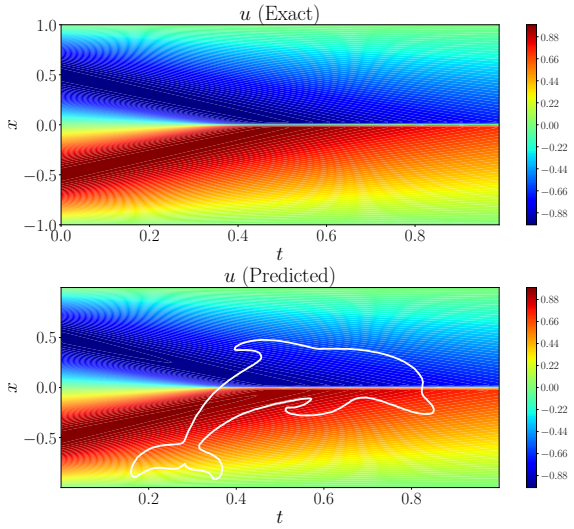

Figure 3: One-dimensional viscous Burgers equation: Contour plot of the exact (left) and predicted (right) solutions over the space-time domain. The white line in the predicted solution represents the dolphin shaped space-time interface.

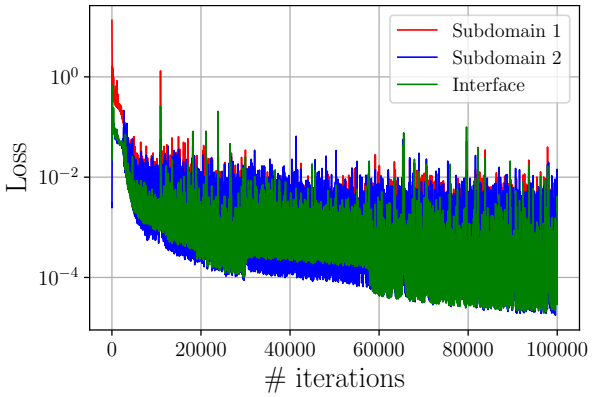

Figure 4: One-dimensional viscous Burgers equation: Loss value for two subdomains as well as for the interface.

Figure 2 shows the location of 300 training data points from initial and boundary conditions. The exact and predicted solutions are shown in the figure 3. After 100k iterations, the relative L_2 error in the solution is 8.93265e-3. Figure 4 shows the convergence history of two subdomains and the interface loss functions, which is given by the following expressions

$$\begin{aligned}
\mathcal{J}(\tilde{\mathbf{\Theta}}_1) &= W_{u_1}\ \text{MSE}_{u_1} + W_{\mathcal{F}_1}\ \text{MSE}_{\mathcal{F}_1}, \text{(Subdomain 1)},\\
\mathcal{J}(\tilde{\mathbf{\Theta}}_2) &= W_{\mathcal{F}_2}\ \text{MSE}_{\mathcal{F}_2}, \qquad\qquad \text{(Subdomain 2)},\\
\mathcal{J}(\tilde{\mathbf{\Theta}}_{\text{Interface}}) &= W_{I_{\mathcal{F}_1}}\ \text{MSE}_{\mathcal{R}} + W_{I_1}\text{MSE}_{u_{avg}}, \text{(Interface)}.
\end{aligned}$$

From the figure we see that both the subdomain and interface losses converge together due to residual continuity condition. We can also observe that initially the subdomain 2 loss is very small due to unavailability of the training data points from the actual computational boundary, thus, it completely depends on the subdomain 1 for the convergence. As the subdomain 1 loss started converging, the subdomain 2 follows the same path, i.e., we see a sudden increase of the loss value and then the convergence. The interface loss is converging faster than subdomains loss and is still decreasing even after 100k iterations. The relative L_2 error in the predicted solution along the interface is 5.92672e-3.

Conclusions

We have proposed a generalized domain decomposition approach, namely the eXtended PINN (XPINN) method. Like the PINN method, the proposed method can be employed to solve any differential equation. This is achieved by enforcing the residual continuity condition along the common interfaces of neighboring subdomains. The residual continuity condition can theoretically enforce the solution regularity across the interface in the classical sense, i.e., the solution across the interface is sufficiently smooth such that it satisfies its governing PDE. We also enforce the average solution (C^0 solution continuity) given by two different neural networks along the common interface between two subdomains, which can increase the convergence rate. The XPINN has all the advantages of its predecessor, the conservative PINN (cPINN) method like deployment of separate neural network in each subdomain, efficient hyper-parameter adjustment (like depth, width, activation function, penalizing points, optimization method etc) for all networks, parallelization capacity, large representation capacity etc. Moreover, a key factor in the XPINN formulation is the flexibility in the division of subdomains for any type of differential equations. The major advantage of the XPINN is that it can be easily employed for any complex simulations involving complex domains, especially in higher dimensions. Overall, the proposed XPINN method is the generalization of PINN and cPINN approaches, both in terms of applicability as well as domain decomposition technique, which efficiently lends itself to parallelized computation.

Acknowledgement

This work was supported by the Department of Energy PhILMs grant DE-SC0019453, the DARPA-AIRA grant HR00111990025, and the DARPA CompMods grant HR00112090062.

References

[1] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, Journal of Machine Learning Research, 18 (2018) 1-43.

[2] C. Basdevant, et al., Spectral and finite difference solution of the Burgers equation, Comput. Fluids, 14 (1986) 23-41.

[3] A.D. Jagtap, Y. Shin, K. Kawaguchi, G.E. Karniadakis, Kronecker neural networks: A general framework for neural networks with adaptive activation functions (In preparation).

[4] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural network: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys., 378, 686-707, 2019.

[5] K. Shukla, P. C. D. Leoni, J. Blackshire, D. Sparkman, G.E. Karniadakis, Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks, arXiv:2005.03596v1, 2020.

[6] D. P. Kingma, J. L. Ba, ADAM: A method for stochastic optimization, arXiv:1412.6980v9, 2017.

[7] A.D. Jagtap and G.E. Karniadakis, Extended physics-informed neural networks (XPINNs) : A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Communications in Computational Physics, 28(5), 2002-2041 (2020).

[8] A.D. Jagtap, K. Kawaguchi and G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys., 404 (2020) 109136.

[9] A.D. Jagtap, K. Kawaguchi and G.E. Karniadakis, Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks, Proc. R. Soc. A 476: 20200334 (2020). http://dx.doi.org/10.1098/rspa.2020.0334

[10] Z. Mao, A.D. Jagtap and G.E. Karniadakis, Physics-informed neural network for high-speed flows, Computer Methods in Applied Mechanics and Engineering, 360 (2020) 112789.

[11] A. D. Jagtap, E. Kharazmi, G.E.Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Computer Methods in Applied Mechanics and Engineering 365 (2020) 113028.