The 9th European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS Congress 2024
3–7 June 2024, Lisboa, Portugal

TOWARDS DOMAIN DECOMPOSITION OF LARGE NEURAL NETWORKS

T. Gödde[1], B. Rosić[1]

[1] Applied mechanics and data analysis (AMDA)
University of Twente
Drienerlolaan 5, 7522NB Enschede, Netherlands
e-mail: t.godde@utwente.nl

Key words: Finite element method, Domain decomposition method, Neural networks

Summary. Neural networks draw attention in many engineering fields. Particularly in design optimization, when one is interested in generating surrogate models for high-dimensional regression problems, neural networks are used. However, neural networks are global approximators and therefore have problems predicting local nonlinearities without a large amount of training parameters slowing down the training process. The idea is to reduce the aforementioned drawbacks by introducing a domain decomposition method. The domain decomposition allows to split global NN into multiple local neural networks to approximate smaller, simpler local domains with less parameters. The simplification comes at the cost of additional interface constraints between local predictions, which are included in the local loss functions by the method of Lagrange multipliers. The method is showcased on a 1D beam example that is split into two domains, which shows that the method works in finding an improved approximation for continuity along the interface constraint.

1 INTRODUCTION

In computational design optimization, the goal is precise knowledge on spatio-temporal Quantities of Interest (QOI) (e.g., stress, strain, etc.) to predict mechanical behaviour. These are typically solved with finite element method (FEM) solvers. However, when an optimization problem is considered, additional dimensions are added to the problem through design parameters. One approach to optimize the design is to take a step into a random direction [1] of the input space and calculate a new FEM simulation online at the new point to check if the new output is better. This is a tedious task and such algorithms take a long time to converge. A more efficient approach is to first map the set of design parameters to the QoI with a surrogate model [2] and utilize the approximation map to find improved design parameters. One example of such an approximation is the use of polynomial chaos expansion (PCE) [3], which can be estimated by the use of a least squares approach [4]. In high dimensional parametric space, this approach requires many different instances of FEM simulations. To reduce the number of required samples, one can use a neural network surrogate [2]. With the appropriate choice of architecture and activation functions, the number of required samples is reduced exponentially [5]. A drawback of this method is that the basis of these neural network surrogate models is a global approximation, which interferes with precision requirements for the local spatio-temporal QoIs. To cope with this loss of information, neural networks are often highly overparametrized, which in turn increases the amount of training time.

Domain decomposition methods (DDM) [6] are used to tackle the previously mentioned drawbacks of a global approximation. DDMs split the spatio-temporal problems in time and space to generate multiple, local problems. Domains are, depending on the method, overlapping [7, 8] or non-overlapping, i.e., sub-structured [9] with the help of a DDM. NN weights, which were previously defined globally, can be now defined locally, on more simple local domains, reducing the amount of unknowns to increase approximation accuracy. Thus, instead of training one large NN, one is training several local NNs in parallel to enable training time reduction. The computational cost reduction, however, comes at the expense of introducing additional unknown parameters into the problem description - those describing the interfaces between local domains. To keep continuity between domains, smooth continuous connections between local domains need to be enforced.

In literature, neural network DDMS are found largely for physics-informed neural networks (PINN) [10]. These are networks in which the loss function is modified to solve boundary value problems, i.e., partial differential equations (PDE) with constraints. Specifically, the loss function is defined as a sum of mean squared error (sMSE) residuals of the problems equations. One of these methods is the deep domain decomposition method (D3M) [11, 12], which solves an overlapping DDM problem. The first D3M is based on the deep Ritz method [13] and solves variational forms of PDEs with local sMSE residual loss functions. Interface conditions are included in the loss functions as additional sMSE penalty terms. Another D3M method uses instead the basic PINN formulation, i.e., strong formulation of a PDE and defines it as local sMSE residual [12] with penalty terms for the interfaces. Another overlapping DDM is the finite-basis PINN (FBPINN) [14]. In FBPINNs, local NN approximations are multiplied by window functions. These window functions ensure that the local network outputs are zero outside the overlap between the domains. Instead of with additional sMSE interface terms, continuity is ensured, since the local NNs share their solutions. As such, in an update step, the local NN gradients are calculated from the global loss function. All previously presented methods require overlap between the domains, which causes additional computational effort in the domain. To cope with this problem, the first substructuring DDMs for PINNs are introduced as extended PINN (XPINN) and conservative PINN (cPINN) [15, 16]. Both of these use penalty based interface conditions on the local loss functions. For continuity in these substructuring methods, XPINNs and cPINNs use additional sMSE penalty terms, resembling the function gradient along the interface and an average between approximation losses, in the local loss functions. The difference between the two is that XPINNs are less limited in chosen PDEs and interface conditions. A remaining drawback of all these methods is that a single local step is followed by a single global step for communication between the networks. However, the global step is computationally expensive. A computationally cheaper approach is to take multiple local steps before communicating between local domains. Recently, this is done for an overlapping DDM method [17]. Two different DDMs are proposed. One of the methods is similar to the previous methods, taking interface constraints as mean squared error (MSE) residual terms, only taking multiple step in the local domains before communicating between the domains. To improve the interface predictions, the second method uses a coarse space for all interfaces connecting them, to increase stability of the DDM. [18] introduces a substructuring method with this coarse space for the interfaces. In addition to that, augmented Lagrange multipliers in front of the residual terms are used in the loss function to improve convergence of the method by dynamically balancing the residual terms.

In this article, a different approach to the NN DDM is presented for parallel computing of different spatial domains. The training data is obtained from a finite element solver to use data-driven NNs. Instead of the augmented Lagrange multipliers, full Lagrange multipliers are used. This gives a more exact solution since the Lagrange multipliers can become any value instead of being kept small due to the penalty term applied for augmented Lagrange multipliers. In addition to that, no penalty parameter is needed with full Lagrange multipliers. The cost is an increase in complexity of the algorithm since the full Lagrange multiplier is harder to solve. With a full Lagrange multiplier, the constraint needs to be linearized, which means linearizing the NN for interface approximations.

The paper is organized as follows: In Section 2 the finite element equations are derived and surrogate models are introduced. The DDM with interface conditions is state in Section 3. Finally, in Section 5 numerical results for a one-dimensional problem are shown and discussed.

2 PROBLEM STATEMENTn

Let there be a body in an Euclidean space $\mathbb{R}^n$, occupying spatial domain $\mathcal{G}$ in which a spatial point of the body is denoted as x. The body has piece-wise smooth Lipschitz boundaries of the Dirichlet $\partial\mathcal{G}_D$ and Neumann $\partial\mathcal{G}_N$ type. Assuming quasi-static deformations and $\zeta \in \mathbb{R}^m$ as the set of design parameters, the following boundary value problem can be stated:

$$\begin{aligned} & & \nabla\cdot\sigma(\zeta,x) &= f(x), & \forall x &\in \mathcal{G}, & (1)\\ &\text{s.t.} & u(x) &= u_D, & \forall x &\in \partial\mathcal{G}_D, & (2)\\ &\text{s.t.} & \sigma(\zeta(\omega),x)\cdot n &= \tau_N, & \forall x &\in \partial\mathcal{G}_N. & (3)\end{aligned}$$

in which σ is the stress tensor, $u \in \mathcal{U}$ with $\mathcal{U} := \mathcal{H}_0^1(\mathcal{G})$ is the displacement field, $f \in \mathcal{U}$ describes the internal forces, τ_n describes a Neumann boundary $\partial\mathcal{G}_N$ and u_D is a Dirichlet boundary $\partial\mathcal{G}_D$. The body is made of material undergoing deformation according to Hooke's constitutive law:

$$\sigma = C : \epsilon, \tag{4}$$

in which $C := C \in \mathcal{L}(Sym(\mathbb{R}^+))$ denotes the 4th order symmetric, bonded, measurable and point-wise stable elasticity tensor C, and the linear strains $\epsilon := \epsilon \in L_2(\mathcal{G}, Sym(\mathbb{R}^+))$ are defined as:

$$\epsilon = \frac{1}{2}(\nabla u + (\nabla u)^T). \tag{5}$$

The denoted strong formulation given in Eqs. (1 - 3) is further rewritten in a weak form as:

$$a(u,v) := \int_{\mathcal{G}} C\epsilon(u) : \epsilon(v)dx = \int_{\mathcal{G}} f(x)\cdot v(x)dx + \int_{\partial\mathcal{G}_N} \tau_n(x)\cdot v(x)dA =: l(v), \tag{6}$$

in which $a(u,v)$ denotes the bi-linear form, and $l(v)$ is the corresponding linear functional. The problem has a unique solution according to the Lax-Milgram theorem [19].

Subsequently, let the design parameters ζ be described in a probabilistic sense. In other words, let ζ be modelled as a random vector $\zeta(\omega)$ with a finite variance in a probability space $(S) := L_2(\Omega, \mathcal{F}, \mathbb{P})$ in which Ω is the space of all events, $\mathcal{F}$ is the sigma algebra, and $\mathbb{P}$ is the probability measure. Introducing $\zeta(\omega)$ in Eqs. (1-3) rewrites the deterministic problem into its stochastic counterpart:

$$\nabla \cdot \sigma(\zeta(\omega), x) = f(x, \omega), \qquad \forall x \in \mathcal{G}, \forall \omega \in \Omega, \tag{7}$$

$$\text{s.t.} \qquad u(x, \omega) = u_D, \qquad \forall x \in \partial\mathcal{G}_D, \forall \omega \in \Omega, \tag{8}$$

$$\text{s.t.} \qquad \sigma(\zeta(\omega), x) \cdot n = \tau_N, \qquad \forall x \in \partial\mathcal{G}_N, \forall \omega \in \Omega. \tag{9}$$

in which $u \in \mathcal{U} \otimes (S)$. The constitutive law in Eq. (4) becomes $\sigma(\omega) = C : \epsilon(\omega)$ with $\epsilon(\omega)$ defined as:

$$\epsilon(\omega) = \frac{1}{2}(\nabla u(\omega) + (\nabla u(\omega))^T), \qquad \forall \omega \in \Omega. \tag{10}$$

Note, that one introduces the weak definition of the linear mapping ∇_s between the displacement $\epsilon \in \epsilon \otimes (S)$ and $u \in \mathcal{U} \otimes (S)$ such that:

$$\nabla_s : u_1(x)u_2(\omega) \to (\nabla_s u_1(x))u_2(\omega), \tag{11}$$

holds. From the stochastic strong form, and similar to Eq. (6), one can derive a corresponding weak form:

$$\int_\Omega \int_\mathcal{G} C\epsilon(u(\omega))) : \epsilon(v(\omega))dxd\Omega = \int_\Omega \int_\mathcal{G} f(x, \omega) \cdot v(x, \omega)dxd\Omega + \int_\Omega \int_{\partial\mathcal{G}_N} \tau_n(x) \cdot v(x, \omega)dAd\Omega, \tag{12}$$

by integrating the terms over Ω. As the solution of this initial value problem belongs to the tensor product space $\mathcal{U} \otimes (S)$, one requires the discretization of each of the spaces separately. First, the deterministic space $\mathcal{U}$ is discretized by using the finite element approach:

$$u_n := span\{N_j(x)\}_{j=1}^{L_n}, \tag{13}$$

such that:

$$u(x, \omega) \approx u_n(x, \omega) = \sum_{i=1}^{L_n} u_i(\omega)N_i(x), \tag{14}$$

where N_i denotes the shape functions and $u_i(\omega)$ represents the unknown stochastic coefficient. Secondly, the stochastic space is discretized by sampling with the Monte Carlo method [1]. Hence, the problem given in Eqs. (12) is solved N times in parallel.

As the Monte Carlo method is not sample efficient, one may introduce a surrogate model to map the input parameters of the parameters to the QoI. In particular, the focus is on a simple architecture in the form of a feed forward neural network (FNN). In other words, the QoI is approximated by the FNN:

$$\boldsymbol{u}(\boldsymbol{x}) \approx \hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x}) = (A_L \circ \boldsymbol{a} \circ A_{L-1} \circ ... \circ \boldsymbol{a} \circ A_1)(\boldsymbol{x}), \qquad \boldsymbol{\theta} := \left\{W^{(k)}, b^{(k)}\right\}_{k=1}^{L}, \tag{15}$$

in which $A_k : \mathbb{R}^{N_{k-1}} \to \mathbb{R}^{N_k}$ is an affine map, with $N_k \in \mathbb{N}$, defined as:

$$x^{(k)} \mapsto W^{(k)}x^{(k-1)} + b^{(k)}. \tag{16}$$

Here, $W^{(k)}$ and $b^{(k)}$ are the trainable parameters collected in the vector $\boldsymbol{\theta}$. In Eq. (15), the vector-valued $\boldsymbol{a}$ represents functions:

$$\boldsymbol{a}(\boldsymbol{h}) = [a(h_1), a(h_2), ..., a(h_P)], \tag{17}$$

acting element-wise on the hidden state h_i of the FNN. P and L denote the amount of nodes per layer and the amount of layers, respectively. To find the unknown parameters $\boldsymbol{\theta}$, one may solve the following optimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}).$$

Here, the objective function $\mathcal{J}$ is typically defined in a mean squared manner:

$$\mathcal{J}(\boldsymbol{\theta}) := \mathbb{E}(\|\boldsymbol{u}(\boldsymbol{x}) - \hat{\boldsymbol{u}}_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2), \tag{18}$$

in which, the expectation is approximated in a Monte Carlo simulation manner, by using a finite number of samples. Finally, the approximation problem is solved by using a gradient-based optimizer [20].

Getting a feasible NN surrogate model approximation is difficult due to the complexity of the previously stated problem. The complexity leads to a high amount of required training parameters, local errors and long training times. Therefore, instead of computing the NN surrogate model in Eq. (15) as a global approximation, the proposed strategy is to reduce the computational requirements by splitting the input features. It is expected, that the local problems are simpler to solve, require less parameters and are more precise locally.

3 DOMAIN DECOMPOSITION METHOD FOR LARGE NNs

Let the spatial domain $\mathcal{G}$ be split into several non-overlapping subdomains $\mathcal{G}_i$, such that $\mathcal{G} = \cup \sum_{i=1}^{M} \mathcal{G}_i$. Thus, the idea is to use the set of M NNs to approximate $\boldsymbol{u}(\boldsymbol{x})$, such that:

$$\boldsymbol{u}_i(\boldsymbol{x}) \approx \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x}) = (A_{L_i}^{(i)} \circ \boldsymbol{a}^{(i)} \circ A_{L_i-1}^{(i)} \circ ... \circ \boldsymbol{a}^{(i)} \circ A_1^{(i)})(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \mathcal{G}_i, \tag{19}$$

with

$$\boldsymbol{a}^{(i)}(\boldsymbol{y}) = [a(y_1), a(y_2), ..., a(y_{P_i})], \tag{20}$$

in which L_i and P_i are the the amount of nodes per layer and the amount of layers of the i^{th} NN for each subdomain, respectively. In Eq. (19), function vector $\boldsymbol{a}^{(i)}$ acts element-wise on hidden states y_i of each subdomain. Given the local NN, the goal is to estimate the unknown parameters $\boldsymbol{\theta}_i$, $i = 1, ..., M$, given the local set of data $(\boldsymbol{x}, \boldsymbol{u}_i(\boldsymbol{x}))$, $\boldsymbol{x} \in \mathcal{G}_i$. To find the optimal parameters, similar to Eq. (18), a mean squared objective of type can be defined as:

$$\boldsymbol{\theta}_i^* = \arg\min \mathcal{J}_i(\boldsymbol{\theta}_i), \quad \mathcal{J}_i(\boldsymbol{\theta}_i) = \mathbb{E}(\|\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \boldsymbol{u}(\boldsymbol{x})\|_2^2), \quad \forall \boldsymbol{x} \in \mathcal{G}_i, \quad i, ..., M, \tag{21}$$

can be used.

However, the previously discussed local approximations suffer from discontinuities at the interfaces between subdomains. To overcome this issue, interfaces between domains must be constrained for a continuous approximation. Therefore, both gradient and function approximation must be the same at the interface for adjacent local NN approximations. Considering first order continuity, the constraints:

$$\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x}) = \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_j}(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ij}, \tag{22}$$

$$\partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x})|_{\boldsymbol{x}} = \partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_j}(\boldsymbol{x})|_{\boldsymbol{x}}, \quad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ij}, \tag{23}$$

must hold. Here, $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}$ is the interface solution defined on the interface $\mathit{\Gamma}_{ik} = \mathcal{G}_i \cap \mathcal{G}_k$, $i \neq k$, and possibly described by its own NN, for parallel computations.

By introducing the interface constraints in Eqs. (22-23) to the system, Eq. (18) rewrites as:

$$\boldsymbol{\theta}_i = \arg\min \mathcal{J}_i, \qquad \mathcal{J}_i(\boldsymbol{\theta}_i) = \mathbb{E}\|\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \boldsymbol{u}(\boldsymbol{x})\|_2^2, \qquad \forall \boldsymbol{x} \in \mathcal{G}_i, \quad i = 1, ..., M, \tag{24}$$

$$s.t. \qquad \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x}) = \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ik}, \tag{25}$$

$$\partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x})|_{\boldsymbol{x}} = \partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}(\boldsymbol{x})|_{\boldsymbol{x}}, \qquad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ik}. \tag{26}$$

On the interface, the objective for $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}$ is defined as:

$$\boldsymbol{\theta}_{ik}^* = \arg\min \mathcal{J}_{ik}(\boldsymbol{\theta}_{ik}), \quad \mathcal{J}_{ik}(\boldsymbol{\theta_{ik}}) = \sum_{j=\{i,k\}} \mathbb{E}(\|\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_j}(\boldsymbol{x}) - \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}(\boldsymbol{x})\|) + \mathbb{E}(\|\partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_j}(\boldsymbol{x})|_{\boldsymbol{x}} - \partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}(\boldsymbol{x})|_{\boldsymbol{x}}\|), \quad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ik}, \tag{27}$$

for two adjacent domains $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}$ and $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_k}$. For simplicity, in the further text one considers only interfaces between two domains.

Numerical optimization requires reformulating Eqs. (24-26) as unconstrained optimisation problem by introducing Lagrange multipliers $\boldsymbol{\lambda}_i$:

$$\mathcal{L}_i(\boldsymbol{\theta}_i, \boldsymbol{\lambda}_i) = \mathcal{J}_i(\boldsymbol{\theta}_i) + \boldsymbol{\lambda}_i^T \boldsymbol{Q}(\boldsymbol{\theta}_i), \tag{28}$$

with

$$\boldsymbol{Q}(\boldsymbol{\theta}_i) := \begin{bmatrix} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}(\boldsymbol{x}) = 0 \\ \partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x})|_{\boldsymbol{x}} - \partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}(\boldsymbol{x})|_{\boldsymbol{x}} = 0 \end{bmatrix}, \qquad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ik}. \tag{29}$$

The resulting optimisation problem is nonlinear, and can not be solved by alternating minimisation [21]. Therefore, one has to linearize the constraints in Eq. (29) around the linearization point $\hat{\boldsymbol{\theta}}_i$ by using a Taylor expansion:

$$\bar{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x}) := \bar{\boldsymbol{u}}(\boldsymbol{\theta}_i, \boldsymbol{x}) = \hat{\boldsymbol{u}}_{\hat{\boldsymbol{\theta}}_i}(\boldsymbol{x}) + \partial_{\boldsymbol{\theta}_i} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x})|_{\hat{\boldsymbol{\theta}}_i}(\boldsymbol{\theta}_i - \hat{\boldsymbol{\theta}}_i), \qquad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ik}, \tag{30}$$

and similar for its derivative

$$\partial_{\boldsymbol{x}} \bar{\boldsymbol{u}}_{\boldsymbol{\theta}_i}|_{\boldsymbol{x}} := \partial_{\boldsymbol{x}} \bar{\boldsymbol{u}}(\boldsymbol{\theta}_i, \boldsymbol{x})|_{\boldsymbol{x}} = \partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\hat{\boldsymbol{\theta}}_i}(\boldsymbol{x})|_{\boldsymbol{x}} + \partial_{\boldsymbol{x}\boldsymbol{\theta}_i} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x})|_{\boldsymbol{x}, \hat{\boldsymbol{\theta}}_i}(\boldsymbol{\theta}_i - \hat{\boldsymbol{\theta}}_i), \qquad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ik}. \tag{31}$$

Substituting Eqs. (30-31) into Eqs. (28-29), one obtains:

$$\bar{\mathcal{L}}_i(\boldsymbol{\theta}_i, \boldsymbol{\lambda}_i) = \mathcal{J}_i(\boldsymbol{\theta}_i) + \boldsymbol{\lambda}_i^T \bar{\boldsymbol{Q}}(\boldsymbol{\theta}_i), \tag{32}$$

with

$$\bar{\boldsymbol{Q}}(\boldsymbol{\theta}_i) = \begin{bmatrix} \bar{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}(\boldsymbol{x}) \\ \partial_{\boldsymbol{x}} \bar{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(\boldsymbol{x})|_{\boldsymbol{x}} - \partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}(\boldsymbol{x})|_{\boldsymbol{x}} \end{bmatrix}, \qquad \forall \boldsymbol{x} \in \mathit{\Gamma}_{ik}. \tag{33}$$

The numerical algorithm for solving the previously stated optimization problem is described in the further text.

4 DUAL DECOMPOSITION ALGORITHM

The algorithm given in Alg. 1, is an alternating minimization algorithm [21]. It consists of two main loops, an outer constraint loop and an inner local loop.

Completion of the constraint loop (2-11) is achieved, when both the Lagrange multipliers $\boldsymbol{\lambda}_i$ and training parameters $\boldsymbol{\theta}_i$ are converged. In the constraint loop, the first step is the communication step between the network approximations in which interface approximation $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}$ is updated. After that, the local loop is performed inside the constraint loop, followed by a step in the Lagrange multipliers based on a gradient-based optimizer.

Termination of the local loop (4-9) is triggered when the training parameters $\boldsymbol{\theta}_i$ converge. The local loop starts with a linearization of the local constraints around the current training parameters $\boldsymbol{\theta}_i$ according to Eqs. (32-33). After that, it performs multiple line searches with the limited-memory Broyden Fletcher Goldfarb Shanno (L-BFGS) optimizer [22] to update the training parameters. After each line search the values of Eq. (29) and Eq. (33) are compared to check if the linearization fails. Only if the linearization fails, a new linearization for the updated weights is performed. Otherwise, the same linearization is taken for multiple line searches.

Algorithm 1 Neural network dual decomposition

```
1: $\boldsymbol{\theta}_i \sim HeNormal$, $\boldsymbol{\lambda}_i = \mathbf{0}$, $\alpha = 1 \cdot 10^{-4}$, $\alpha_i$ defined by Hager-Zhang line search
2: while $\frac{1}{N_{\boldsymbol{\lambda}_i}} \|\lambda_{i+1} - \lambda_i\|_2^2 > 0$ do
3:     Update interface approximation $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{ik}}$
4:     while $\mathcal{L}_i^{u+1} - \mathcal{L}_i^u > 0$ do
5:         Compute $\bar{\boldsymbol{Q}}(\boldsymbol{\theta}_i)$ to linearize $\bar{\mathcal{L}}_i(\boldsymbol{\theta}_i, \boldsymbol{\lambda}_i) = \frac{1}{N_{\boldsymbol{x}_i}} \|\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i} - \boldsymbol{u}\|_{2(\boldsymbol{x}_i)}^2 + \boldsymbol{\lambda}_i^T \bar{\boldsymbol{Q}}(\boldsymbol{\theta}_i)$
6:         while $\frac{1}{N_C} \|\boldsymbol{Q}(\boldsymbol{\theta}_i^k) - \bar{\boldsymbol{Q}}(\boldsymbol{\theta}_i^k)\|_2^2 = 0$ do
7:             Approximate $\frac{\partial \bar{\mathcal{L}}_i}{\partial \boldsymbol{\theta}_i} \approxeq \boldsymbol{p}$ for new direction by L-BFGS
8:             Take line search step $\boldsymbol{\theta}_i^k = \boldsymbol{\theta}_i^{k-1} - \alpha_i \boldsymbol{p}$
9:         end while
10:    end while
11:    Update Lagrange multipliers $\boldsymbol{\lambda}_i^k = \boldsymbol{\lambda}_i^{k-1} + \alpha \boldsymbol{Q}(\boldsymbol{\theta}_i^k)$
12: end while
```

5 Results

Let be given the steel beam in Figure 1a that is clamped on the left side. A vertical point load of $1000N$ is applied on its right end. The steel beam's geometric and material properties can be found in Table 1. Discretized into 1000 linear beam elements, the beam is simulated with the FEM to get the vertical displacement field of the beam. This results in 1001 nodal positions and displacements along the beam, which are split into two datasets. At the intersection of the two datasets, as shown schematically in Figure 1b, some of the information from the FEM simulation is neglected to resemble a gap of 60 data points (6 mm) between the two approximations. Consequently, each local dataset consists of 441 equally distant training points. Two NNs $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_1}(\boldsymbol{x})$ and $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_2}(\boldsymbol{x})$ are generated, with their properties shown in Tab, to approximate the vertical displacement fields along the beam for the two datasets. 2. In Figure 1b, the interface point Γ_{12} is denoted as a large dot. On the interface point, four Lagrange multipliers are defined, two for each local domain, for the constraints in displacements and displacement gradients. It is assumed that prediction $\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_{12}}(x_{12})$ on the interface is equal to $(\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_1}(x_{12}) + \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_2}(x_{12}))/2$ for $x_{12} \in \mathit{\Gamma}_{12}$ and similar for the displacement gradients, since the solution in 1D is very simple. Another important preparation step is to normalize the input and output values on the interval (-1,1). This is trivial for most of the data. Special care is only taken when normalizing the displacement gradient constraints given in Eq. (29), which is normalized by setting:

$$\partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}^n_{\boldsymbol{\theta}_i} = \frac{2}{\pi} \arctan\left(\partial_{\boldsymbol{x}} \hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}\right). \tag{34}$$

This approach utilizes trigonometry to bound the gradient $\partial_{\boldsymbol{x}} \boldsymbol{u}^n_{\boldsymbol{\theta}_i}$ on the interval $(-1, 1)$.

To look into the effectiveness of the proposed algorithm, its convergence is analysed. Figure 2 shows the loss of Eq. (28) for domains $D1$ and $D2$. The graphs show the training loss in logarithmic scale over the iterations of the algorithm. In both figures, it is observed that the loss converges to a fixed value quickly after the first iteration. In addition to that, the final value of the losses is slightly higher than the loss after the first iteration. This can be explained by the fact that the Lagrange multipliers are initialized at zero and updated after the first iteration. At this point, the constraints are introduced to the loss value, increasing it. After that point, a trade-off between the domain approximation and the error at the interface must be made in the following iterations. It can be seen that after 5-10 iterations the loss stabilizes to a fixed value in both figures, showing convergence of the algorithm.

The exact effect of including the interface constraint in the loss function is shown in Figures 3. In these, Figures 3a and 3b show the converged approximations of the two networks defined in domains D1 and D2 with and without the interface constraint. In each graph, two approximation lines for domains D1 and D2, and two vertical lines, which indicate the region without data points, can be observed. In addition to that, black dots are shown to indicate the data from the FEM data set. The improvement by the algorithm on the interface is clear. While on Figure 3a, the two NN approximations have a small discontinuity and their gradients are different, 3b

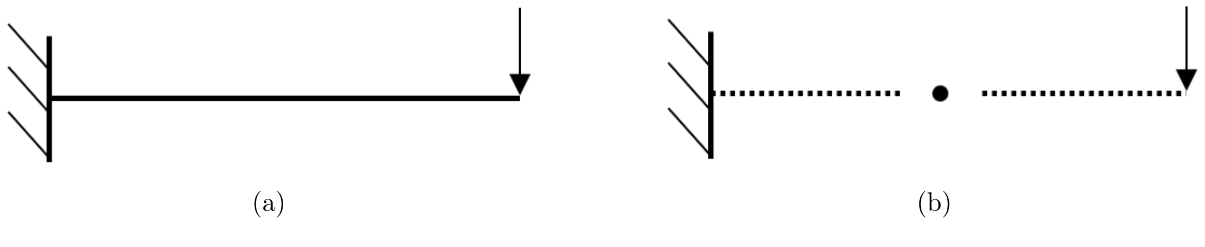


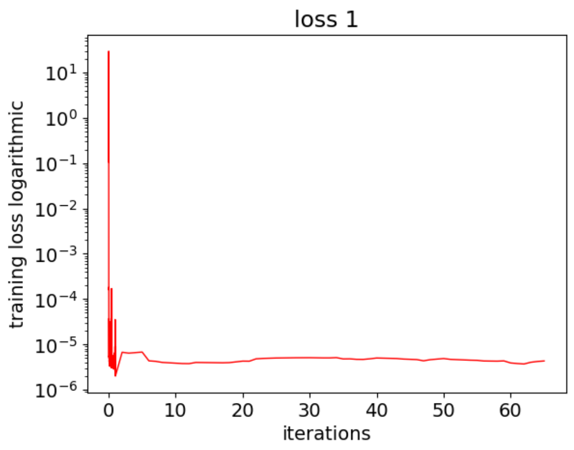
Figure 1: a) schematically shows a clamped beam that is loaded vertically. b) schematically shows the beam from (a) as discretized points with a gap in the middle and a point as interface.

Table 1: Beam properties for the finite element simulation.

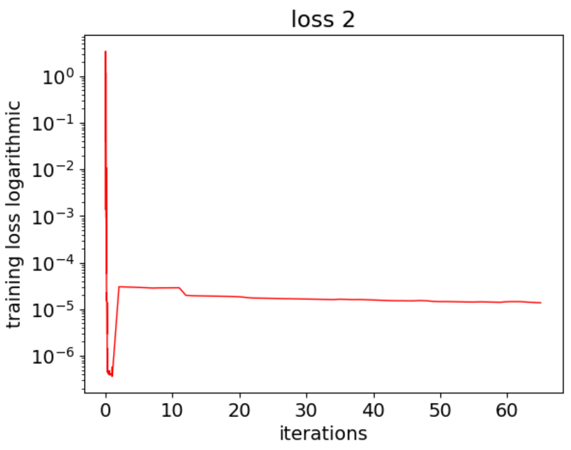
Dimensions [mm^3]					Young's	Poisson	Element	Number of	
width	x	height	x	length	mod [MPa]	ratio	type	elements	nodes
10	x	10	x	100	21000	0.3	linear beam	1000	1001

Table 2: Neural network parameters.

Network	Architecture			Activation	Optimizer	Batch	Initializer
name	width	x	depth	function		size	
$\hat{\boldsymbol{u}}_{\theta_1}$	10	x	2	swish	L-BFGS	441	He Normal
$\hat{\boldsymbol{u}}_{\theta_2}$	10	x	2	swish	L-BFGS	441	He Normal



(a) Convergence of the loss function in domain zero.



(b) Convergence of the loss function in domain one.

Figure 2: Loss functions during operation of the algorithm. The first iteration shows all until convergence while the later steps only show converged final loss values.

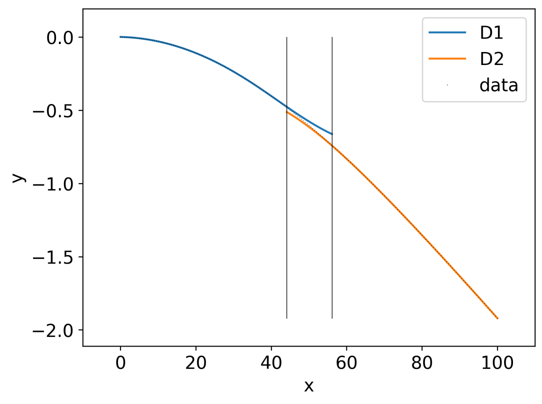
indicates that the NN approximations are on top of each other with the same gradient on the interface point. The relative error Figures 3c and 3d zoom in on this observation. In these, the relative error is defined as:

$$e_{rel} = \frac{|\hat{\boldsymbol{u}}_{\boldsymbol{\theta}_i}(x) - \boldsymbol{u}(x)|}{|\boldsymbol{u}(x)|+1},$$

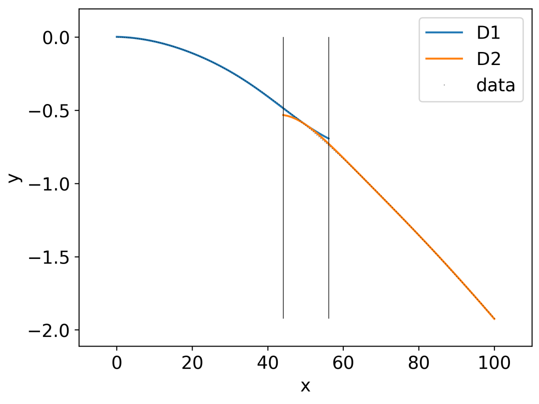
which uses a modified relative error to cope with the zero value NN approximations. In the figure, the relative errors of each NN approximation, denoted as domains D1 and D2, are shown up to the interface. A clear discontinuity shows near the interface before the algorithm (Fig. 3c) that is then greatly reduced and made continuous after the algorithm (Fig. 3d). Without the interface constraint, the highest relative error is 1.75% and the gradients are in opposite directions. After convergence, it can be seen that the slopes are aligned and the predictions are very close to each other, with an error of less than 0.2%. At the same time, the highest error reduced by a factor of around 2.5, while the local predictions are barely influenced by the interface modification.

6 Conclusion

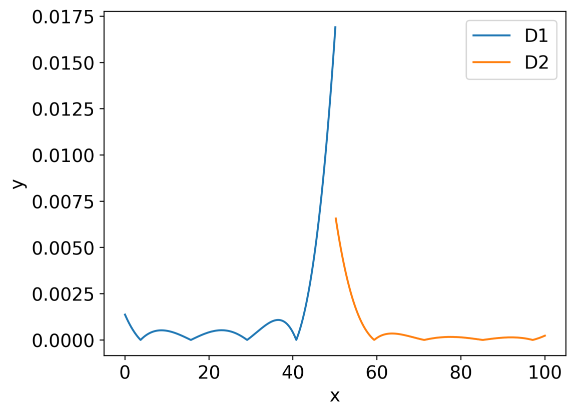
A DDM to split a global NN approximation into simpler local NN approximations is presented. The local NNs are updated by a local loss functions using gradient-based solvers. To keep a continuous prediction, interface constraints are introduced on the local losses. The interface constraints are enforced by Lagrange multiplier, resulting in an unconstrained problem, which is done by an alternating minimization according to Algorithm 1. The method is presented on a 1D beam problem in which the beam is split into two approximation domains. It is found that the method greatly reduces the interface discontinuity, to smoothly connect the two local domains. Given the promising results, in the future, this method will be extended to multiple dimensions and more complex problems. In addition to that, theoretical proofs for its convergence need to be elaborated.



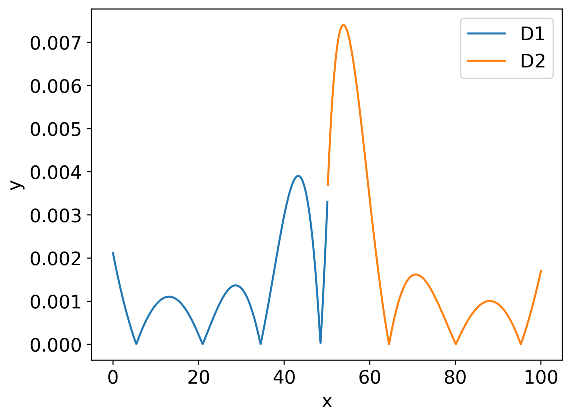
(a) Predictions without interface constraint.



(b) Predictions with interface constraint.



(c) Relative error without interface constraint.



(d) Relative error with interface constraint.

Figure 3: Neural network predictions before and after the algorithm. The Blue lines are predictions and relative errors of D1 and the orange lines the predictions and relative errors of D2.

References

[1] V. Chan. *Theory and Applications of Monte Carlo Simulations.* Rijeka: IntechOpen, Mar. 2013. ISBN: 9789535110125. DOI: 10.5772/45892.

[2] A. Forrester, A. Sobester, and A. Keane. *Engineering Design via Surrogate Modelling.* John Wiley Sons, Ltd, 2008. Chap. 4, pp. 109–116. ISBN: 9780470770801.

[3] N. Wiener. "The Homogeneous Chaos". In: *American Journal of Mathematics* 60.4 (Oct. 1938), pp. 897–936. URL: http://www.jstor.org/stable/2371268 (visited on 08/14/2024).

[4] T. Zhou, A. Narayan, and D. Xiu. "Weighted discrete least-squares polynomial approximation using randomized quadratures". In: *Journal of Computational Physics* 298 (Oct. 2015), pp. 787–800. DOI: https://doi.org/10.1016/j.jcp.2015.06.042.

[5] A. R. Barron. "Universal approximation bounds for superpositions of a sigmoidal function". In: *IEEE Transactions on Information Theory* 39.3 (1993), pp. 930–945. DOI: 10.1109/18.256500.

[6] A. Toselli and O. Widlund. *Domain Decomposition Methods - Algorithms and Theory.* Vol. 34. Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer, 2005. ISBN: 978-3-540-20696-5. DOI: 10.1007/b137868. URL: https://doi.org/10.1007/b137868.

[7] H.A. Schwarz. "Über einige Abbildungsaufgaben". In: *Journal für die reine und angewandte Mathematik* 1869.70 (1869), pp. 105–120. DOI: doi:10.1515/crll.1869.70.105.

[8] P. L. Lions. "On the Schwarz Alternating Method I". In: *First Proceedings of Domain Decomposition Methods for Partial Differential Equations.* Ed. by R. Glowinski, G.H. Golub, G.A. Meurant, and J. Periaux. Philadelphia, USA: SIAM, 1988. Chap. 1, pp. 1–42.

[9] P. L. Lions. "On the Schwarz Alternating Method III: A Variant for Nonoverlapping Subdomains". In: *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations.* Ed. by T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund. Philadelphia, USA: SIAM, 1990. Chap. 11, pp. 202–223.

[10] 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". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. DOI: https://doi.org/10.1016/j.jcp.2018.10.045.

[11] K. Li, K. Tang, T. Wu, and Q. Liao. "D3M: A Deep Domain Decomposition Method for Partial Differential Equations". In: *IEEE Access* 8 (Dec. 2019), pp. 5283–5294. DOI: 10.1109/ACCESS.2019.2957200.

[12] W. Li, X. Xiang, and Y. Xu. "Deep Domain Decomposition Method: Elliptic Problems". In: *Proceedings of The First Mathematical and Scientific Machine Learning Conference.* Ed. by L. Jianfeng and W. Rachel. Vol. 107. PMLR, 2020, pp. 269–286.

[13] M. Liu, Z. Cai, and K. Ramani. "Deep Ritz method with adaptive quadrature for linear elasticity". In: *Computer Methods in Applied Mechanics and Engineering* 415 (2023), p. 116229. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2023.116229.

[14] B. Moseley, A. Markham, and T. Nissen-Meyer. "Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations". In: *Adv Comput Math* 49 (2023), pp. 1–39. DOI: https://doi.org/10.1007/s10444-023-10065-9.

[15] 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”. In: *CiCP* 28.5 (2020), pp. 2002–2041. DOI: `https://doi.org/10.4208/cicp.OA-2020-0164`.

[16] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis. “Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems”. In: *Computer Methods in Applied Mechanics and Engineering* 365 (2020), p. 113028. ISSN: 0045-7825. DOI: `https://doi.org/10.1016/j.cma.2020.113028`.

[17] H. H. Kim and H. J. Yang. “Domain Decomposition Algorithms for Neural Network Approximation of Partial Differential Equations”. In: *Domain Decomposition Methods in Science and Engineering XXVII*. Ed. by Z. Dostál, T. Kozubek, A. Klawonn, U. Langer, L. F. Pavarino, J. Šístek, and O. B. Widlund. Cham: Springer Nature Switzerland, 2024, pp. 27–37.

[18] K. Jang, K. Kim, and H. H. Kim. “Partitioned neural network approximation for partial differential equations enhanced with Lagrange multipliers and localized loss functions”. In: *CoRR* (2023). DOI: `10.48550/ARXIV.2312.14370`.

[19] H. Brezis. “Compact Operators. Spectral Decomposition of Self-Adjoint Compact Operators”. In: *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. New York, NY: Springer New York, 2011, pp. 157–179. ISBN: 978-0-387-70914-7. DOI: `10.1007/978-0-387-70914-7_6`.

[20] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006.

[21] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. “Distributed optimization and statistical learning via the alternating direction method of multipliers”. In: *Foundations and Trends in Machine Learning* 3 (Jan. 2011), pp. 1–122. DOI: `10.1561/2200000016`.

[22] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. “A Limited Memory Algorithm for Bound Constrained Optimization”. In: *SIAM Journal on Scientific Computing* 16.5 (Feb. 1995), pp. 1190–1208. DOI: `10.1137/0916069`.