Machine Learning in Adaptive FETI-DP – A Comparison of Smart and Random Training Data

Alexander Heinlein, Axel Klawonn, Martin Lanser, and Janine Weber

1 Introduction

The convergence rate of classical domain decomposition methods for diffusion or elasticity problems usually deteriorates when large coefficient jumps occur along or across the interface between subdomains. In fact, the constant in the classical condition number bounds [11, 12] will depend on the coefficient jump. Therefore, several adaptive approaches to enrich the coarse space with additional coarse modes or primal constraints, which are constructed from the solutions of localized eigenvalue problems, have been developed to overcome this limitation, e.g., [7, 6, 13, 14, 2, 3]. For many realistic coefficient distributions, however, only a few adaptive constraints on a few edges or faces are necessary for robustness. Although some heuristic approaches [6, 7] exist to reduce the number of eigenvalue problems that have to be solved, in general, we do not know in advance on which edges or faces additional adaptive constraints are needed to obtain a robust algorithm.

To overcome this issue, we consider an approach to train a neural network to predict the geometric location of adaptive constraints in a preprocessing step, i.e., to make the decision whether or not we have to solve a certain eigenvalue problem. First results using this machine learning based strategy in the context of adaptive domain decomposition methods for a concrete and carefully designed set of training data can be found in [5]. Here, in addition to [5], we test the feasibility of using randomly

Alexander Heinlein, Axel Klawonn, and Martin Lanser
Department of Mathematics and Computer Science, University of Cologne, Weyertal 86-90, 50931 Köln, Germany, e-mail: alexander.heinlein@uni-koeln.de,axel.klawonn@uni-koeln.de,martin.lanser@uni-koeln.de, url: http://www.numerik.uni-koeln.de
Center for Data and Simulation Science, University of Cologne, url: http://www.cds.uni-koeln.de

Janine Weber
Department of Mathematics and Computer Science, University of Cologne, Weyertal 86-90, 50931 Köln, Germany, e-mail: janine.weber@uni-koeln.de, url: http://www.numerik.uni-koeln.de

generated training data for the neural network. Random training data can be easily generated without any knowledge of the considered model problem, and therefore, the approach discussed here is more general compared to [5]; however, a larger set of training data may be required; cf. section 4. We provide numerical results for four different sets of training data to train the neural network and compare the robustness of the resulting algorithms both with respect to the training and validation data as well as for a concrete test problem.

We focus on a stationary diffusion problem in two dimensions and a certain adaptive coarse space technique for the FETI-DP (Finite Element Tearing and Interconnecting - Dual-Primal) algorithm [13, 14]. The adaptive coarse space is implemented using a balancing preconditioner. Let us remark that all strategies introduced here and in [5] can be generalized for arbitrary adaptive domain decomposition methods in two dimensions.

2 Model Problem and Adaptive FETI-DP

As a model problem, we use a stationary diffusion problem in two dimensions with various highly heterogenous coefficient functions $\rho : \Omega := [0,1] \times [0,1] \to \mathbb{R}$, i.e., the weak formulation of

$$\begin{aligned} \operatorname{div}(\rho \nabla u) &= -1 \text{ in } \Omega \\ u &= 0 \text{ on } \partial\Omega. \end{aligned} \tag{1}$$

In this paper, we apply the proposed machine learning based strategy to a certain adaptive FETI-DP method. We thus decompose the domain Ω into $N \in \mathbb{N}$ nonoverlapping subdomains $\Omega_i, i = 1, \ldots, N$. Due to space limitations, we do not explain the standard FETI-DP algorithm in detail. For a detailed description of the FETI-DP algorithm, see, e.g., [12]. Note that we choose all vertices as primal variables.

As already mentioned, for arbitrary and complex coefficient functions ρ, using solely primal vertex constraints is not sufficient to obtain a robust algorithm or condition number bound, respectively. Additional adaptive constraints, resulting from the solution of localized eigenvalue problems, are needed to enrich the coarse space and guarantee robustness. In our case, the adaptive constraints are implemented in FETI-DP by using a balancing preconditioner. For a detailed description of projector or balancing preconditioning, see [10]. In all our computations, we exclusively use ρ-scaling. Please note that also other approaches are possible to enforce coarse constraints, e.g., a transformation of basis approach [12].

The main idea of the concrete adaptive FETI-DP algorithm [13, 14] is to solve a local generalized eigenvalue problem for each edge between two neighboring subdomains. For a detailed description of the specific local edge eigenvalue problem as well as the resulting enforced coarse constraints, see [13, 14]. Usually, it is not known in advance on which edges additional coarse components are necessary. Although the solution of the different eigenvalue problems and thus the computation of the adaptive constraints can be parallelized, building the adaptive coarse space can make up the larger part of the overall time to solution. As already mentioned,

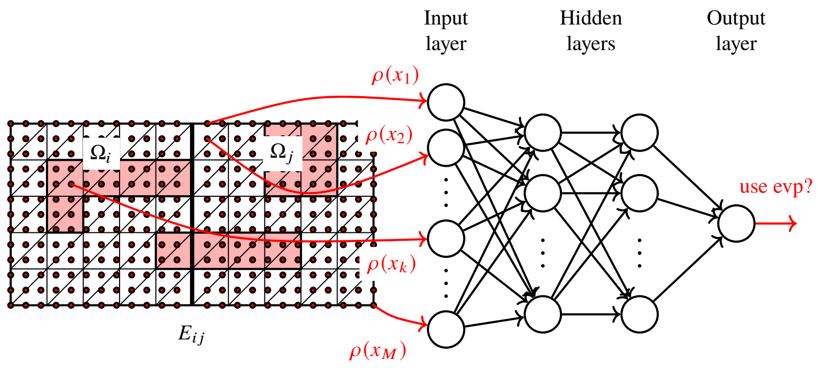


Fig. 1: Sampling of the coefficient function ρ; white color corresponds to a low coefficient and red color to a high coefficient. In this representation, the samples are used as input data for a neural network with two hidden layers. Figure from [5, Fig. 2].

we suggest a machine learning based approach to avoid the solution of unnecessary eigenvalue problems in order to save compute time.

3 Machine Learning for Adaptive FETI-DP

Our approach is to train a neural network to automatically make the decision whether an adaptive constraint needs to be enforced on a specific edge, or not, to retain the robustness of the algorithm, depending on a user-given tolerance TOL.

Supervised machine learning in general approximates nonlinear functions, which associate input and output data. The training of a neural network in supervised machine learning corresponds to the solution of a high-dimensional nonlinear optimization problem. In this paper, we use a dense feedfoward neural network, or more precisely, a multilayer perceptron. For more details on multilayer perceptrons, see, e.g., [15, 1, 16]. As input data for our neural network, we use samples of the coefficient functions within the two subdomains adjacent to an edge; cf. Figure 1. We use a sampling approach which is independent of the finite element discretization. In particular, we use a fixed number of sampling points for all mesh resolutions but assume the sampling grid to resolve all geometric details of the coefficient function. Our sampling grid is oriented to the tangential and orthogonal direction of an edge. Therefore, our approach is also valid for more general subdomain geometries than square subdomains; see also [5]. As output of the neural network, we save the information whether an adaptive coarse constraint has to be computed for the considered edge or not. Our neural network consists of three hidden layers with 30 neurons each. For all hidden layers, we use the ReLU function as an activation function and a dropout rate of 20%. For the training of the neural network, we use the stochastic gradient descent algorithm with an adaptive scaling of the learning rate and a batch size of 100. As an optimizer for the stochastic gradient descent method, we use

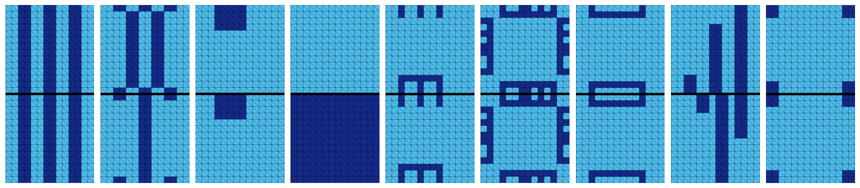


Fig. 2: Nine different types of coefficient functions used for training and validation of the neural network. The inclusions, channels, boxes, and combs with high coefficient are displaced, modified in sized, and mirrored with respect to the edge in order to generate the complete training data set.

the Adam (Adaptive moments) optimizer. All the aforementioned parameters result from applying a grid-search algorithm with cross-validation over a discrete space of hyper-parameters for the neural network; see [5] for more details on the parameters.

For the numerical results presented in this paper, we only train on two regular subdomains sharing a straight edge. Regarding the coefficient functions, we use different sets of coefficient distributions to generate different sets of training data. For the first set of training data, we use a total of 4,500 configurations varying the coefficient distributions as shown in Figure 2. These coefficient distributions are inspired by those used in [8, 2], and all coefficient distributions shown in Figure 2 are varied in size, location, and orientation to obtain the full set of training data. We refer to this set of training data, which already has been used in [5], as *smart data*.

Next, we consider random data to train the neural network. Let us note that a completely random coefficient distribution is not appropriate since in this case coefficient jumps appear at almost all edges. Thus, for almost every edge an eigenvalue problem has to be solved. This yields a neural network which overestimates the number of eigenvalue problems needed and thus leads to a large number of false positive edges in the test data which here is given by the microsection problem.

Thus, as a second set of training data, we use a slightly more structured set of randomly generated coefficients with a varying ratio of high and low coefficient values. For the first part of this training set, we randomly generate the coefficient for each pixel, consisting of two triangular finite elements, independently and only control the ratio of high and low coefficient values. Here, we use 30%, 20%, 10%, and 5% of high coefficient values. For the second part, we also control the distribution of the coefficients to a certain degree by randomly generating either horizontal or vertical stripes of a maximum length of four or eight pixels, respectively; see Figure 3. Additionally, we generate new coefficient distibutions by superimposing pairs of horizontal and vertical coefficient distributions. We refer to this second set of training data as *random data*.

To generate the output data that is necessary to train the neural network, we solve the eigenvalue problems as described in [13, 14] for all the aforementioned training and validation configurations. Here, we basically propose two different classification approaches as already considered in [5]. The first approach is referred to as 'two-class classification' and classifies an edge to belong to class 0 if no adaptive constraint needs to be added to the coarse space for the respective edge, depending on the

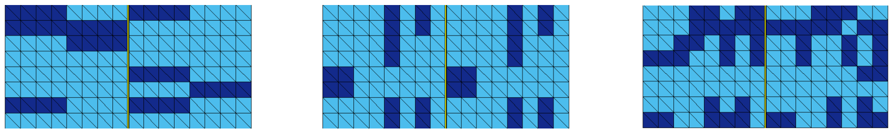


Fig. 3: Examples of three different randomly distributed coefficient functions obtained by using the same randomly generated coefficient for a horizontal (**left**) or vertical (**middle**) stripe of a maximum length of four finite element pixels, as well as by pairwise superimposing (**right**).

user-based tolerance TOL. It is classified to belong to class 1 if at least one adaptive constraint needs to be added. We further provide a second approach, which is referred to as 'three-class classification'. Here, besides class 0, we further distinguish between class 1, for edges where exactly one adaptive constraint needs to be added to the coarse space, and class 2, for edges where more than one constraint is necessary. For class 1, we replace the eigenvalue problem and the resulting eigenvector by a single edge constraint designed using ρ, which therefore also avoids the solution of some eigenvalue problems. These edge constraints can be interpreted as a generalization of the weighted edge averages suggested in [9] and are robust for a broader range of heterogeneities; see [4] for a detailed discussion. For all our training and validation data, we use a tolerance of $TOL = 100$ to generate the output for each edge.

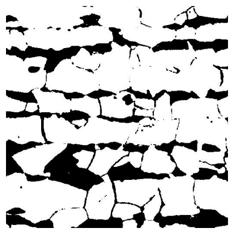
4 Numerical Results

In this section, we compare the performance of the proposed machine learning based adaptive strategy for the FETI-DP algorithm using different sets of training and validation data to train our neural network. In particular, we use a set of 4,500 *smart data* configurations (denoted by *'S'*) and sets of 4,500 and 9,000 *random data* configurations (denoted by *'R1'* and *'R2'*, respectively) each individually as well as a combination of 4,500 *smart* and 4,500 *random data* configurations, which will be denoted by *'SR'*. Note that we did not observe a significant improvement for the larger number of 18,000 random data configurations.

First, we will present results for the whole set of training data using cross-validation and a fixed ratio of 20% as validation data to test the generalization properties of our neural network. Please note that due to different heterogeneity of the various training data, the accuracies in Table 1 are not directly comparable with each other. However, the results in Table 1 serve as a sanity check to prove that the trained model is able to generate appropriate predictions. We will then use 10 different randomly chosen subsections of a microsection of a dual-phase steel as shown in Figure 4 (right) as a test problem for the trained neural network. In all the computations, we consider $\rho = 1e6$ in the black part of the microsection and $\rho = 1$ elsewhere. Here, we use a regular decomposition of the domain Ω into 64 square subdomains, a subdomain size of $H/h = 64$, and a tolerance of $TOL = 100$. Please note, that also other mesh resolutions of the finite element mesh can be

Table 1: Results on the complete training data set; the numbers are averages over all training configurations. We show the ML-threshold (τ), the number of false positives (**fp**), the number of false negatives (**fn**), and the accuracy in the classification (**acc**). We define the accuracy as the number of true positives and true negatives divided by the total number of training configurations.

					class		
training configuration	τ	**fp**	**fn**	**acc**	**0**	**1**	**2**
4,500 smart data S, two-class	0.45	8.8%	1.9%	89.2%	67%	33%	-
	0.5	5.4%	5.1%	89.5%			
4,500 smart data S, three-class	0.4	5.1%	1.0%	93.9%	67%	20%	13%
	0.5	3.2%	2.3%	94.5%			
4,500 random data R1, two-class	0.45	11.4%	6.7%	81.9%	49%	51%	-
	0.5	8.8%	9.0%	82.2%			
4,500 random data R1, three-class	0.4	9.1%	7.1%	83.8%	49%	39%	12%
	0.5	8.9%	7.0%	84.1%			
9,000 random data R2, two-class	0.45	9.6%	5.3%	85.1%	53%	47%	-
	0.5	7.2%	7.5%	85.3%			
9,000 random data R2, three-class	0.4	10.7%	4.4%	84.9%	53%	28%	19%
	0.5	7.4%	6.9%	85.7%			
4,500 smart + 4,500 random data SR, two-class	0.45	5.1%	2.1%	92.8%	58%	42%	-
	0.5	3.4%	3.5%	93.1%			
4,500 smart + 4,500 random data SR, three-class	0.4	5.2%	2.0%	92.8%	58%	29.5%	12.5%
	0.5	4.3%	2.2%	93.5%			



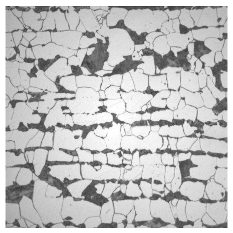


Fig. 4: Left: Subsection of a microsection of a dual-phase steel obtained from the image on the right. We consider $\rho = 1e6$ in the black part and $\rho = 1$ elsewhere. **Right:** Complete microsection of a dual-phase steel. Right image: Courtesy of Jörg Schröder, University of Duisburg-Essen, Germany, orginating from a cooperation with ThyssenKruppSteel.

used without affecting the accuracy of our classification algorithm as long as the coefficient function is constant on each finite element; see also [5]. For the test data, we will only compute the local eigenvalue problems on edges which are classified as critical (class 1 or 2) by the neural network. On all uncritical edges (class 0), we do not enforce any constraints. We use an ML (Machine Learning) threshold τ of 0.5 and 0.45 for the two-class classification as well as 0.5 and 0.4 for the three-class classification, respectively, for the decision boundary between critical and uncritical edges. A lower threshold decreases the false negative rate of the predictions and thus increases the robustness of our algorithm. All computations are performed using the machine learning implementations in TensorFlow and Scikit-learn as well as our Matlab implementation of the adaptive FETI-DP method.

Table 2: Comparison of standard FETI-DP, adaptive FETI-DP, and ML-FETI-DP for **regular domain decompositions** for the **two-class model**, for 10 different subsections of the microsection in Figure 4 (right). Here, training data is denoted as **T-Data**. We show the ML-threshold (τ), the condition number (**cond**), the number of CG iterations (**it**), the number of solved eigenvalue problems (**evp**), the number of false positives (**fp**), the number of false negatives (**fn**), and the accuracy in the classification (**acc**). We show the average values as well as the maximum values (in brackets).

Alg.	**T-Data**	τ	**cond**	**it**	**evp**	**fp**	**fn**	**acc**
standard	-	-	1.5e6 (2.2e6)	>300	0	-	-	-
adaptive	-	-	11.0 (15.9)	34.6 (38)	112.0 (112)	-	-	-
ML	S	0.5	8.6e4 (9.7e4)	39.5 (52)	45.0 (57)	1.6 (2)	1.9 (3)	0.97 (0.96)
	S	0.45	11.0 (15.9)	34.6 (38)	46.9 (59)	**4.4 (6)**	**0 (0)**	0.96 (0.94)
	R1	0.5	1.3e5 (1.6e5)	49.8 (52)	43.2 (44)	7.4 (8)	3.8 (4)	0.88 (0.87)
	R1	0.45	11.0 (15.9)	34.6 (38)	53.8 (58)	14.6 (16)	0 (0)	0.86 (0.84)
	R2	0.5	1.5e5 (1.6e5)	50.2 (51)	40.4 (41)	5.6 (6)	3.4 (4)	0.91 (0.89)
	R2	0.45	11.0 (15.9)	34.6 (38)	50.4 (52)	11.2 (12)	0 (0)	0.90 (0.87)
	SR	0.5	9.6e4 (9.8e4)	45.8 (48)	38.2 (39)	1.8 (2)	1.6 (2)	0.96 (0.95)
	SR	0.45	11.0 (15.9)	34.6 (38)	43.4 (44)	4.8 (5)	0 (0)	0.96 (0.94)

Table 3: Comparison of standard FETI-DP, adaptive FETI-DP, and ML-FETI-DP for **regular domain decompositions** for the **three-class model**, for 10 different subsections of the microsection in Figure 4 (right). Here, training data is denoted as **T-Data**. By **e-avg** we denote the generalized edge average described at the end of Section 3. See Table 2 for the column labeling. We show the average values as well as the maximum values (in brackets).

Alg.	**T-Data**	τ	**cond**	**it**	**evp**	**e-avg**	**fp**	**fn**	**acc**
standard	-	-	1.5e6 (2.2e6)	>300	0	-	-	-	-
adaptive	-	-	11.0 (15.9)	34.6 (38)	112.0 (112)	-	-	-	-
ML	S	0.5	147.4 (271.4)	48.8 (58)	4.2 (10)	43.6 (46)	1.8 (3)	1.4 (3)	0.97 (0.95)
	S	0.4	12.4 (16.4)	34.8 (39)	16.0 (24)	24.2 (28)	10.6 (16)	0 (0)	0.90 (0.85)
	R1	0.5	1.8e4 (1.8e4)	70.4 (72)	7.4 (8)	31.4 (33)	1.8 (2)	1.2 (2)	0.97 (0.94)
	R1	0.4	12.4 (16.4)	34.8 (39)	28.2 (30)	20.4 (26)	16.4 (21)	0 (0)	0.84 (0.83)
	R2	0.5	2.2e4 (2.5e4)	69.4 (72)	5.8 (7)	28.8 (31)	1.6 (2)	1.2 (2)	0.96 (0.95)
	R2	0.4	12.4 (16.4)	34.8 (39)	23.6 (24)	17.0 (18)	15.6 (17)	0 (0)	0.84 (0.83)
	SR	0.5	142.5 (286.3)	52.4 (66)	7.2 (9)	30.6 (32)	1.8 (2)	1.8 (2)	0.96 (0.94)
	SR	0.4	12.4 (16.4)	34.8 (39)	18.2 (20)	16.2 (18)	**10.0 (12)**	**0 (0)**	0.90 (0.89)

Results on the training data: With respect to the training data, the results in terms of accuracy in Table 1 show that, besides training the neural network with the set of smart data, also the training with randomly generated coefficient functions as well as with a combination of both training sets lead to an appropriate model. Thus, it is reasonable to apply all of the trained models to our test problem in form of microsection subsections.

Results on microsection subsections: For the mircosections and the two-class classification, see Table 2, all four different training data sets result in a robust algorithm when using an ML threshold $\tau = 0.45$. For all these approaches, we obtain no false negative edges, which are critical for the convergence of the algorithm. However, the usage of 4,500 and 9,000 random data (see R1 and R2) results in a higher number of false positive edges compared to the sole use of 4,500 smart data, resulting in a larger number of computed eigenvalue problems. Also for the three-class classification, see Table 3, the usage of all four aforementioned training data sets results in zero false negative edges when using the ML threshold $\tau = 0.4$. In this case, we further obtain a quantitatively smaller difference between the training data S and R1 or R2, respectively, in terms of false positive edges than for the two-class classification.

As a conclusion, we observe that we were able to achieve comparable results when using randomly generated coefficient distributions as training data compared to the manually selected smart data; this is beneficial since the random data can be generated without a priori knowledge. However, we need a higher number of random data and a slight structure in the random coefficient distributions to achieve the same accuracy as for the smart data. It also seems possible to slightly improve the performance of the neural network trained using a combination of smart data and random data for the training.

References

1. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning, vol. 1. MIT press Cambridge (2016)
2. Heinlein, A., Klawonn, A., Knepper, J., Rheinbach, O.: Multiscale coarse spaces for overlapping Schwarz methods based on the ACMS space in 2d. Electron. Trans. Numer. Anal. **48**, 156–182 (2018)
3. Heinlein, A., Klawonn, A., Knepper, J., Rheinbach, O.: An adaptive GDSW coarse space for two-level overlapping Schwarz methods in two dimensions. In: Domain Decomposition Methods in Science and Engineering XXIV, *LNCSE*, vol. 125, pp. 373–382. Springer (2019). DOI:10.1007/978-3-319-93873-835
4. Heinlein, A., Klawonn, A., Lanser, M., Weber, J.: A Frugal FETI-DP and BDDC Coarse Space for Heterogeneous Problems (2019). TR series, Center for Data and Simulation Science, University of Cologne, Germany, Vol. 2019-18. https://kups.ub.uni-koeln.de/10363/. Submitted for publication
5. Heinlein, A., Klawonn, A., Lanser, M., Weber, J.: Machine Learning in Adaptive Domain Decomposition Methods - Predicting the Geometric Location of Constraints. SIAM J. Sci. Comput. **41**(6), A3887–A3912 (2019)
6. Klawonn, A., Kühn, M., Rheinbach, O.: Adaptive coarse spaces for FETI-DP in three dimensions. SIAM J. Sci. Comput. **38**(5), A2880–A2911 (2016)
7. Klawonn, A., Kühn, M., Rheinbach, O.: Adaptive FETI-DP and BDDC methods with a generalized transformation of basis for heterogeneous problems. Electron. Trans. Numer. Anal. **49**, 1–27 (2018)
8. Klawonn, A., Radtke, P., Rheinbach, O.: A comparison of adaptive coarse spaces for iterative substructuring in two dimensions. Electron. Trans. Numer. Anal. **45**, 75–106 (2016)
9. Klawonn, A., Rheinbach, O.: Robust FETI-DP methods for heterogeneous three dimensional elasticity problems. Comput. Methods Appl. Mech. Engrg. **196**(8), 1400–1414 (2007)
10. Klawonn, A., Rheinbach, O.: Deflation, projector preconditioning, and balancing in iterative substructuring methods: connections and new results. SIAM J. Sci. Comput. **34**(1), A459–A484 (2012)
11. Klawonn, A., Rheinbach, O., Widlund, O.B.: An analysis of a FETI-DP algorithm on irregular subdomains in the plane. SIAM J. Numer. Anal. **46**(5), 2484–2504 (2008)
12. Klawonn, A., Widlund, O.B.: Dual-primal FETI methods for linear elasticity. Comm. Pure Appl. Math. **59**(11), 1523–1572 (2006)

13. Mandel, J., Sousedík, B.: Adaptive selection of face coarse degrees of freedom in the BDDC and the FETI-DP iterative substructuring methods. Comput. Methods Appl. Mech. Engrg. **196**(8), 1389–1399 (2007)
14. Mandel, J., Sousedík, B., Sístek, J.: Adaptive BDDC in three dimensions. Math. Comput. Simulation **82**(10), 1812–1831 (2012)
15. Müller, A., Guido, S.: Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media (2016)
16. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning. Cambridge University Press (2014)